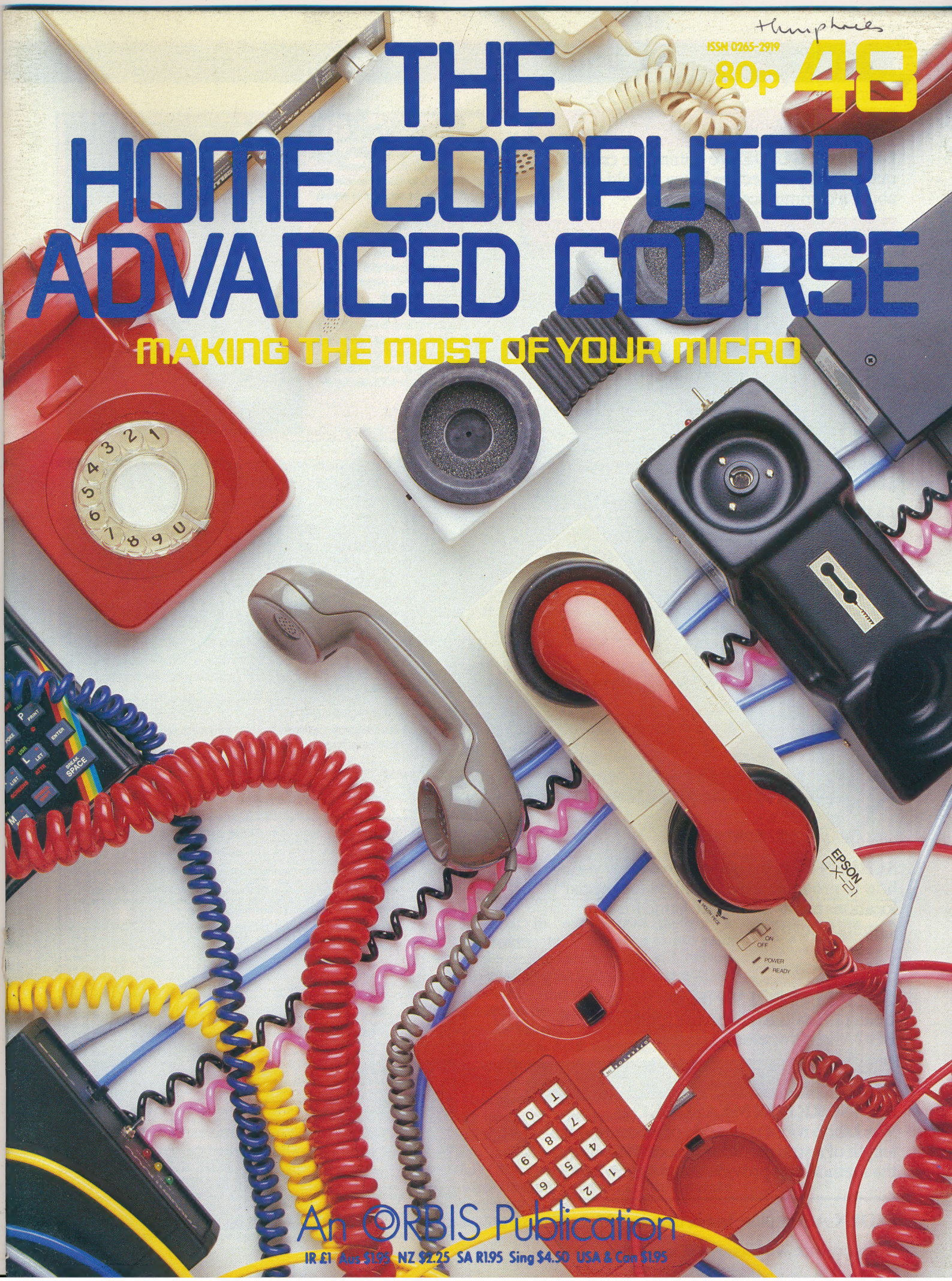
ISSN 0265-2919

80p 48

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



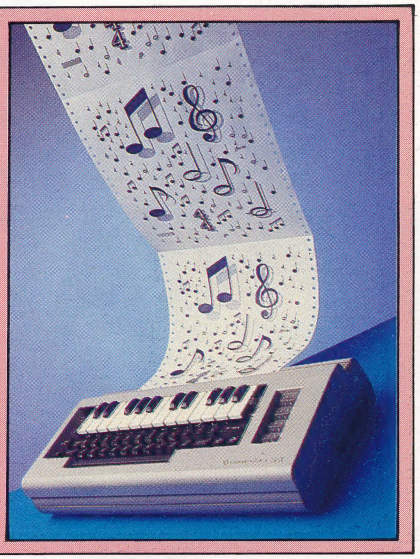
An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **The Music Maker is a music package with a keyboard overlay for the Commodore 64. We try our hand at making music with it.**
- **Few computers are designed for children, but My Talking Computer is a machine aimed at 3- to 10-year-olds. We consider its educational and entertainment value.**
- **Continuing our Workshop series, we design a program to allow the robot to scan an area and determine the shape of an object within the area.**



QUIZ

1) When measuring the length of an object with our Workshop robot, what feature of the sensors will limit the accuracy of the measurement?
2) Does the Human Edge package give an objective assessment of a counterpart from data provided?
3) What effect does prefixing a BBC command with a * have?
4) What is a 'silicon disk'?

Answers To Last Week's Quiz

1) A 'software board' is a touch-sensitive peripheral available for the Sharp PC-1500A.
2) Tom West was the project leader in the development of the Eagle minicomputer.
3) Data has to be detokenised because the target computer may not use the same set of tokens as the host computer.

Editor Mike Wesley; **Art Editor** Claudia Zeff; **Technical Editor** Brian Morris; **Production Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Staff Writer** Stephen Malone; **Consultant Editor** Steve Colwill; **Contributors** Geoff Bains, Harvey Mellar, Joe Pritchard, Steve Malone, Karl Dallas, Surya, Steve Colwill, Andrew Bangham; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Manager** Peter Taylor-Medhurst; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA –** Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders. **UK/EIRE** – Price: 80p/IR£1. Subscription: 6 months: £23.92. 1 Year: £47.84. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 80p. Subscription: 6 months air: £43.68. Surface: £34.84. 1 year air: £87.36. Surface: £69.68. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 80p. Subscription: 6 months air: £45.76. Surface: £34.84. 1 year air: £91.52. Surface: £69.68. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CANS1.95/80p. Subscription: 6 months air: £54.08. Surface: £34.84. 1 year air: £108.16. Surface: £69.68. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R1.95. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 80p. Subscription: 6 months air: £58.24. Surface: £34.84. 1 year air £116.48. Surface: £69.68. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: AusS1.95. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZS2.25. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY IAN McKINNELL TELEPHONES PROVIDED COURTESY OF THE TELEPHONE BOX

BUYERS' GUIDE

Having taken a detailed look at how computers communicate with each other, we now turn our attention to the vexed problem of choosing a suitable modem and communications package for your micro. Rather than suggesting specific packages, however, we outline some of the more pertinent questions you can ask.

You may have been advised, when originally selecting your computer system, that you should first decide what software you want to use and then choose the hardware that will run it. To some degree, this is sound advice when choosing a communications system. However, we suggest that it is often far better to buy your modem and software as a package from the same dealer.

The first thing you must do is decide what you're going to use the system for. To access viewdata systems such as Prestel you'll need a 1200/75 baud modem; for most bulletin boards and electronic mail systems you'll need a 300 baud modem; and for direct user-to-user work, a 1200 baud rate is recommended. To access Compunet, a special Compunet modem is required, since some of the software is stored in the modem's ROM. You'll also need to take into account the frequencies used. In the UK, you require CCITT frequencies; in the US, you need Bell tones. Therefore, if you intend to make direct transatlantic calls, your modem needs to be capable of both frequencies.

For user-to-user communications, it's strongly advisable to have a modem that can be switched between 'originate' (transmission) and 'answer' (reception) frequencies. If your modem operates only on originate, the modem you're calling must be capable of being switched to answer.

If you intend to call bulletin boards, an *auto-dial* modem is virtually essential. This is because most bulletin boards have only a single telephone line and allow access to one user at a time. For this reason they are frequently engaged, so it makes sense to get the modem to do the repeated dialling for you. An auto-dial modem should also have supporting software that is capable of obtaining a number — either from the keyboard or from a database of phone numbers — and sending this to the modem in the correct format. Unfortunately, different auto-dial modems want the number sent to them in different forms, so the modem and software need to be compatible — a good argument for buying both modem and software as one package.

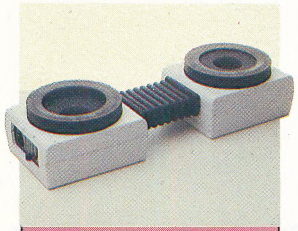
If you want people to send data directly to you, you may find an *auto-answer* modem a worthwhile investment. If you plan to use your normal telephone line for this purpose, however, it's polite to warn your friends — particularly those without modems. Otherwise, your modem may whistle at them for ten seconds, and then hang up!

Appropriate software is also essential for an auto-answer modem. Such software ranges from packages capable of opening a new file for each call and saving this to disk, to sophisticated bulletin board software such as TBBS. An interesting piece of auto-answer software for CP/M micros is Remote CP/M. This package allows you to dial up your CP/M micro and execute any CP/M program over the telephone — which is ideal for users with both a desk-top and a portable micro.

In choosing suitable software, you will almost

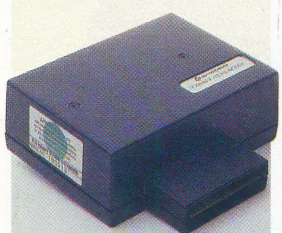
STEVE CROSS

Making The Connection
The range of available hardware and software options makes buying a modem a confusing and specialised task: our ideal unit (shown here) combines the features of a number of actual modems. For a beginner, the best approach is to specify your anticipated communications requirements, and rely on a dealer to choose the appropriate package

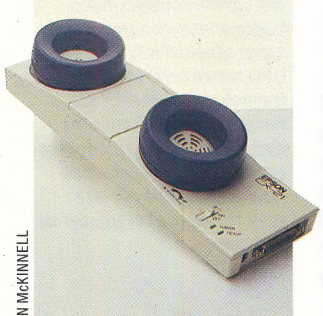


Protek 1200
Type: Acoustic modem
Baud Rate: 1200/75 and 1200/1200
Price: £99.95

Prism 1000
Type: Hard-wired Viewdata modem
Baud Rate: 1200/75
Price: Modem only: £69.95; Full machine-specific package: £129.95



Commodore Communications Modem
Type: Hard-wired Viewdata modem for Compunet
Baud Rate: 1200/75 and 1200/1200
Price: £99.99



IAN McKINNELL

Epson CX-21
Type: Acoustic modem
Baud Rate: 300
Price: £185

certainly want a package that is able to up-load and down-load ASCII files. Make sure the package supports whatever storage device you use. BASIC programs can be transmitted in ASCII form — as we've already shown (see page 921) — but if you want to transmit binary files (for example, CP/M .COM files), you'll need some kind of binary transmission protocol. Of these, the most widely supported is XModem.

It's also convenient to be able to create *auto-log on* files for different systems. Then, when you log on to a system, all you have to do is load the appropriate file, containing your ID, password and so on. Some auto-dial systems will link this type of file to a database of phone numbers so that all you need do is enter the name of the service you want; the software will look up the phone number, dial it and automatically log on.

Once you've decided on the features you require, you need to find a modem and software package that supports these facilities. You can buy the modem and software separately, but we strongly recommend that you give a dealer a list of the features you require, and details of the micro you will be using, and leave it to him to find a complete package of modem, cable and software. That way, if the system doesn't do what you want it to, you and the dealer both know whose responsibility it is to put it right.

CHOOSING A SUITABLE TERMINAL

If you already have a micro you'll probably want to use it as your terminal. This should be possible whatever machine you have — even a ZX81 can be used if you're determined enough — although some micros are better suited to communications applications than others. Here is a brief summary of the suitability of four of the most popular micros

By far the easiest machine to convert to a terminal is the BBC Micro. In fact, you can write a simple dumb terminal program for it in a few lines of BASIC:

```
100 REM BBC Dumb Terminal Program
110 *FX2,2
120 *FX3,1
130 REPEAT: GET A$: IF$=CHR$(13) THEN PRINT
140 PRINT A$;:UNTIL FALSE
```

Several good communications packages are available for the BBC Micro, some of which are supplied on ROM, but they tend to be expensive. Most offer all the features you need, because the BBC Micro's operating system does most of the work — all the programmer has to do is add finishing touches.

The Spectrum is more difficult to adapt. Firstly, you won't be able to achieve any communications breakthrough in BASIC. With a BASIC program it's just about possible to push the Spectrum up to about 10 baud, and then it won't be able to perform such tasks as storing the characters in RAM. You can also disregard your Interface 1: it's not an RS232 interface and is little use for communications. Virtually no communications software is available for the Spectrum yet.

The Commodore 64 also has a non-standard serial interface, and most modems for the machine plug into the user port. Again, you can't do anything very useful in BASIC. The 64 also has a non-standard ASCII character set, so the communications software needs to translate between standard ASCII and Commodore ASCII — something that can be done easily using a look-up table. There are no Commodore-approved communications software packages. UK users can obtain Termulator from Chris Townsend Computers; US users should check with local dealers. Compunet users can use the official Commodore Compunet modem with resident software, but this modem can be used only for Compunet.

Tandy supplies dumb terminal software for its disk-based TRS-80 machines running under most operating systems. Most of it is intended primarily for direct machine-to-machine transfer but can also be used with modems. Tandy machines were the first micros used to run and access bulletin boards, so there is normally a good selection of both disk- and cassette-based public-domain communications software to be found. TRS-80s are not suitable for viewdata (1200/75 baud) operation.

Almost any business micro can be used for communications, but there are several important points to check. Firstly, there's a growing tendency towards built-in modems; these — particularly the ones with ROM-based communications software — are obviously the easiest to use. They simply need to be plugged into the telephone socket.

Failing a built-in modem, the next best option is a micro with at least two RS232 ports, allowing you to use a modem and serial printer simultaneously. Some micros have separate, non-standard modem ports: these will serve, assuming you can get a suitable cable, for your chosen modem.

In terms of software, you shouldn't have problems with a CP/M, MS-DOS or PC-DOS machine. Non-standard operating systems, however, are just as much a liability with communications packages as with any other software.

If communications is your main reason for buying a micro, the so-called 'lap-held' machines such as the Tandy Model 100, the NEC PC8201A and the Olivetti M10 are well worth considering. With one of these and a battery-operated modem you have a conveniently portable briefcase terminal. All three machines have built-in text editors and terminal software and give about 20 hours use from four AA battery cells. These machines retail at between £300 and £500, and a portable acoustic modem will cost you from £180 to £250.

MEASURE FOR MEASURE

We return to our Workshop robot project to design a piece of software that will allow the robot to locate and accurately measure one side of a straight-sided object.

To allow our robot to locate and measure the side of an object requires a reasonably sophisticated piece of software. The robot will probe the object using the microswitch sensors that we fitted on page 876. Our first thoughts about a possible method of accomplishing this task were:

1) Find the object.
2) Find one end of the side located.
3) Probe along the side of the object until the other end is met.

The first stage can be accomplished easily if we assume that when the program starts the robot is pointed at the side of the object we want to measure. The main problem that can be foreseen is that the robot may catch one end of the side with a single sensor rather than make contact with both sensors. The possible variations are shown in the diagram. However, even if only one front sensor is closed, it is possible to tell whether it is the left or right sensor, and, therefore, we can develop a strategy to deal with this situation.

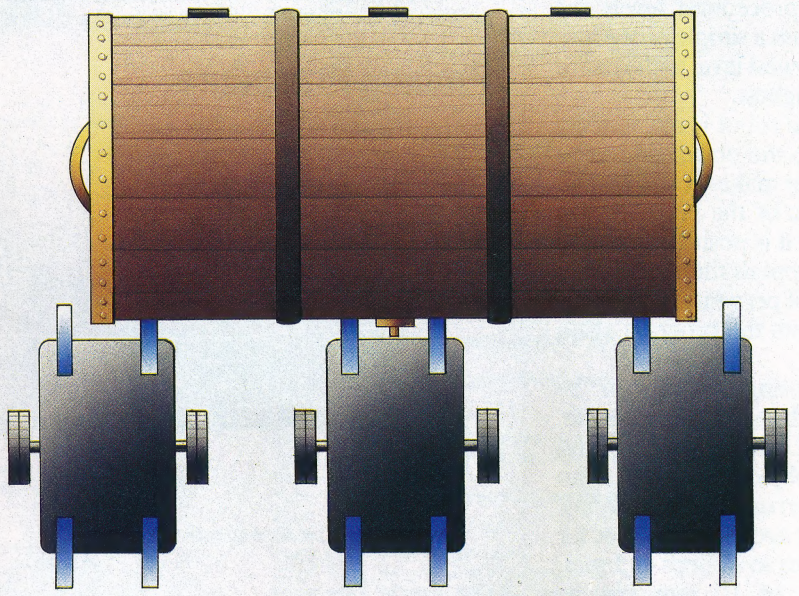
We must also make the assumption that the robot is initially positioned at 90° to the object side to be measured. This avoids our having to deal with cases where the robot makes an oblique contact with the object.

The second stage in our method is simplified so that the robot always moves to the right-hand end of the object before starting to measure it. To locate the right-hand end, the robot must 'feel' its way down the side, moving in discrete steps to the right until only the left sensor (rather than both) closes. In order to probe down the side, the robot has to perform a complicated series of manoeuvres — each step involving five movements. Assuming that the robot is initially in contact with the side of the object, then it must reverse back, turn through 90°, move forward a certain distance, turn back 90°, and finally move forward until the sensors again make contact with the object. The diagram shows the steps involved in the full manoeuvre. The 'step-length' (the distance between one contact point with the object's side and the next) is equivalent to the part of the manoeuvre when the robot moves parallel to the object's side.

To locate accurately the right-hand end of the object, it would appear that the robot would have to probe along the side in steps of a few millimetres, but this is unnecessary. Instead, we can use larger steps, probing down the side of the

Feelings . . .

This diagram shows the three alternatives that could occur when the sensors come into contact with the side of an object. When only the right-hand sensor is closed, the robot has detected the left-hand end of the side; if both are closed, it 'knows' that it's somewhere in the middle; if only the left-hand sensor is closed, it has stumbled on the right-hand end of the side



Right sensor only closed | Both sensors closed | Left sensor only closed

KEVIN JONES

object until the end is overshot, and then backtrack one step to probe in smaller steps to locate the right-hand end of the object accurately. A suitable step-length is the distance between the two front sensors — around 60 mm ($2\frac{3}{8}$ in) — since this ensures that when an overshoot occurs, one sensor will still close.

The third stage involves a similar method of probing, but this time the robot moves to the left, counting the number of steps taken until the left-hand end is encountered. At the end of this stage, the length of the object side will be held in the count variable and can be PRINTed.

MEASUREMENT PROGRAM

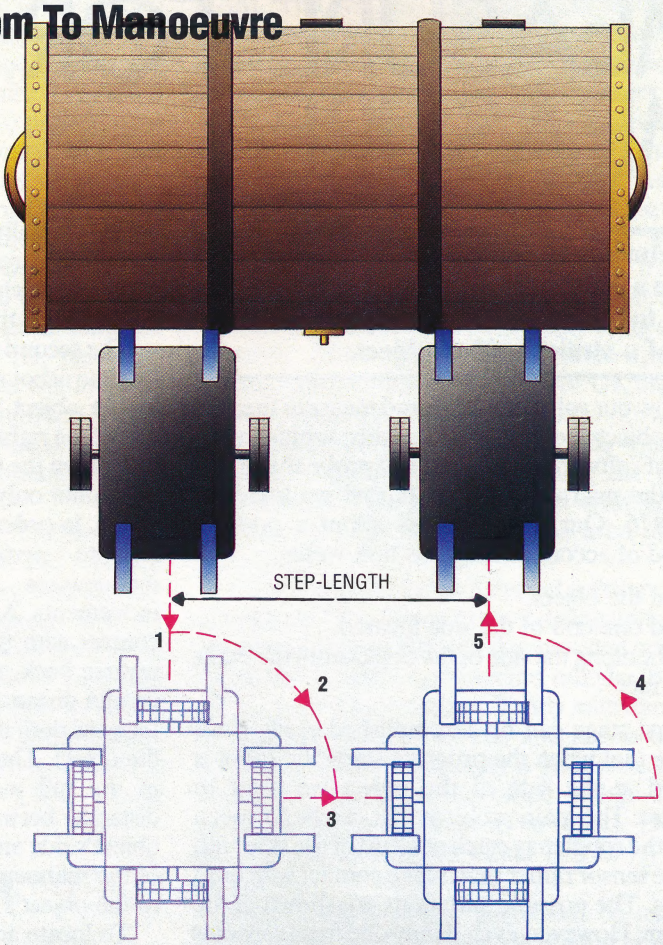
We give listings for the Commodore 64 and BBC Micro. The pulse/distance and pulse/angle ratios — found by experiment in the last instalment of the robot project (see page 894) — should be inserted for your own robot. The procedural nature of BBC BASIC is ideal for writing a program of this type. We can adopt a highly structured approach to the problem, controlling each movement made by the robot via separate procedures. Two features of BBC BASIC — extended variable names and parameter-passing between procedures — mean that the program can more closely resemble the way we think. The Commodore version can adopt the same structured approach, but notice how much more difficult structuring is in Commodore BASIC — the Commodore 64 program is much more difficult to follow than the BBC version.

Having isolated the main tasks that the program must perform, we can design individual procedures for moving the robot and combining a series of manoeuvres to create a 'probing' procedure. Combining probing procedures forms the overall 'measure' procedure. In this application, the different procedural levels are easily identified, ranging from a simple procedure to pulse the motors at the lowest level to the entire measuring activity at the highest.

Problems can arise if the robot is not initially positioned at exactly 90° to the object side to be measured. If only one sensor makes contact when both should, then the logic of the program will make the robot decide that it is positioned at one end of the object. If this happens then break into the program, align the robot perpendicular to the side to be measured, and run the program again from the beginning.

Several intrinsic measuring errors can be identified. The width of each sensor, for example, is around 5 mm ($\frac{3}{16}$ in). Locating the left- and right-hand ends of the object, therefore, can produce a total maximum error of 10 mm ($\frac{3}{8}$ in). In addition, when the robot accurately probes for the ends of the object, it does so in steps of 5 mm, and thus a further error of 10 mm can be introduced. When testing our prototype robot, the average error in measuring an object of side 410 mm (16 in) was around 20 mm ($\frac{3}{4}$ in) — an error of only five per cent.

Room To Manoeuvre

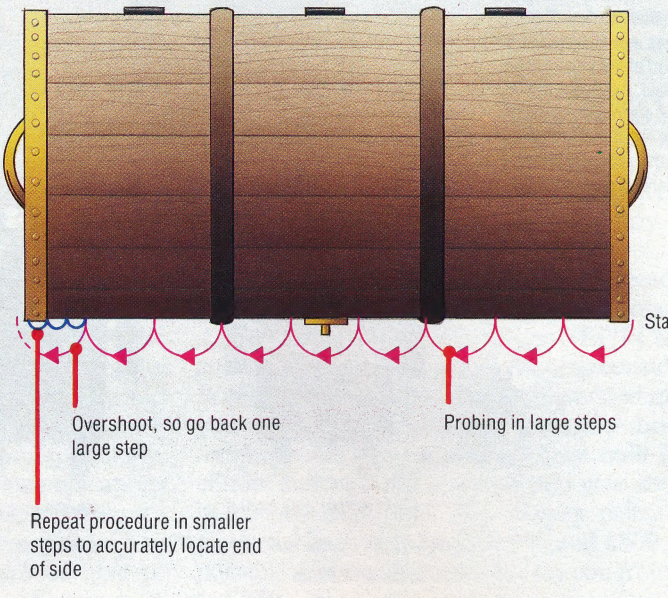


The figure above demonstrates the robot's basic probing manoeuvre. When both sensors are closed, the robot will reverse (1) to enable it to turn without colliding with the object.

The robot will then perform a 90° turn (2) and move forward the step-length (3). The robot performs another 90° turn so that it is once more facing the object (4), and finally moves forward (5) to test whether the object is still in front of it.

This process is repeated along the length of the object, as shown below. At first the robot takes a number of large steps, to gain an approximate length of the object. When only one, or neither, of the sensors is closed, the robot backtracks the last step, and repeats the whole probing procedure using smaller steps

Probing Procedure



KEVIN JONES

BBC Micro Listing

```
1000 REM **** BBC ROBOT MEASURE ****
1010 MODE 7
1020 PROCinitialise
1030 PROCmeasure
1040 PROCprintout
1050 END
1060 DEF PROCmeasure
1070 PROCfind
1080 REM ** ONE BUMPER ONLY ? **
1090 PROCtest_bumpers
1100 REM **** FIND END ****
1110 REPEAT:PROCprobe(right,width)
1120 UNTIL (?DATREG AND 192)=left_bumper
1130 REM ** GO BACK AND INCH TO END **
1140 PROCprobe(left,width)
1150 REPEAT:PROCprobe(right,small_width)
1160 UNTIL (?DATREG AND 192)=left_bumper
1170 PRINT"FOUND RIGHT-HAND END"
1180 PRINT"STARTING TO MEASURE"
1190 REM ** START TO MEASURE **
1200 count=width
1210 REPEAT:PROCprobe(left,width)
1220 count=count+width
1230 UNTIL (?DATREG AND 192)=right_bumper
1240 REM ** GO BACK AND INCH TO END **
1250 count=count-width
1260 PROCprobe(right,width)
1270 REPEAT:PROCprobe(left,small_width)
1280 count=count+small_width
1290 UNTIL (?DATREG AND 192)=right_bumper
1300 ?DATREG=0
1310 ENDPROC
1320 :
1330 DEF PROCprintout
1340 CLS
1350 PRINTTAB(5,12)"OBJECT SIDE MEASURED AT
     ";count;" mm"
1360 ENDPROC
1370 :
1380 DEF PROCinitialise
1390 DDR=&FE62:DATREG=&FE60
1400 ?DDR=15:REM LINES 0-3 OUTPUT
1410 ?DATREG=1:REM TURN ON RESET BIT
1420 forwards=4:backwards=2:left=6:right=0
1430 pd_ratio=3.34446:pa_ratio=375/90
1440 right_bumper=128:left_bumper=64
1450 both_bumpers=0:neither_bumpers=192
1460 width=60:small_width=5
1470 ENDPROC
1480 :
1490 DEF PROCsearch(sense)
1500 REPEAT:PROCprobe(sense,width)
1510 UNTIL (?DATREG AND 192)=both_bumpers
1520 ENDPROC
1530 :
1540 DEF PROCfind
1550 REPEAT:PROCmove(forwards,8)
1560 UNTIL(?DATREG AND 192)<>neither_bumpers
1570 ENDPROC
1580 :
1590 DEF PROCtest_bumpers
1600 IF (?DATREG AND 192)=right_bumper THEN
     PROCsearch(right):ENDPROC
1610 IF (?DATREG AND 192)=left_bumper THEN
     PROCsearch(left):ENDPROC
1620 ENDPROC
1630 :
1640 DEF PROCprobe(way,step)
1650 IF way=right THEN opp_way=left ELSE
     opp_way=right
1660 PROCmove(backwards,30)
1670 PROCturn(way,90)
1680 PROCmove(forwards,step)
1690 PROCturn(opp_way,90)
1700 REPEAT:PROCmove(forwards,8)
1710 UNTIL (?DATREG AND 192)<>neither_bumpers
1720 ENDPROC
1730 :
1740 DEF PROCmove(dir,distance)
1750 ?DATREG=(?DATREG AND 1)OR dir
1760 pulses=pd_ratio*distance
1770 FOR I=1 TO pulses:PROCpulse:NEXT I
1780 ENDPROC
1790 :
1800 DEF PROCturn(dir,angle)
1810 ?DATREG=(?DATREG AND 1)OR dir
1820 pulses=pa_ratio*angle
1830 FOR I=1 TO pulses:PROCpulse:NEXT I
1840 ENDPROC
1850 DEF PROCpulse
1860 ?DATREG=(?DATREG OR 8)
1870 ?DATREG=(?DATREG AND 247)
1880 ENDPROC
```

Commodore 64 Listing

```
10 REM **** CBM ROBOT MEASURE ****
20 GOSUB1000:REM INITIALISE
30 GOSUB2000:REM MEASURE
40 GOSUB3000:REM PRINTOUT
50 END
60 :
1000 REM **** INITIALISE ****
1010 DDR=56579:DATREG=56577
1020 POKE DDR,15:REM LINES 0-3 OUTPUT
1030 POKE DATREG,1:REM TURN ON RESET BIT
1040 FW=4:BW=2:LF=6:RT=0
1050 PD=3.34446:PA=375/90
1060 RB=128:LB=64:BB=0:NB=192
1070 WD=60:SW=5
1080 RETURN
1090 :
2000 REM **** MEASURE ****
2010 GOSUB3500:REM FIND OBJECT
2020 GOSUB4000:REM TEST BUMPERS
2030 REM ** FIND END **
2040 WY=RT:SP=WD:GOSUB6000:REM PROBE
2050 IF(PEEK(DATREG)AND192)<>LB THEN 2040
2060 REM ** GO BACK AND INCH TO END **
2070 DR=LF:DS=WD:GOSUB6000:REM PROBE
2080 DR=RT:DS=SW:GOSUB6000:REM PROBE
2090 IF(PEEK(DATREG)AND192)<>LB THEN 2080
2100 PRINT"FOUND RIGHT-HAND END"
2110 PRINT"STARTING TO MEASURE"
2120 REM ** START TO MEASURE **
2130 CC=WD
2140 DR=LF:DS=WD:GOSUB6000:CC=CC+WD
2150 IF(PEEK(DATREG)AND192)<>RB THEN 2140
2160 REM ** GO BACK AND INCH TO END **
2170 CC=CC-WD
2180 DR=RT:DS=WD:GOSUB6000:CC=CC+SW
2190 IF(PEEK(DATREG)AND192)<>RB THEN 2180
2200 POKE DATREG,0
2210 RETURN
2220 :
3000 REM **** PRINT OUT ****
3010 PRINTCHR$(147)
3020 PRINT"OBJECT MEASURED AT ";CC;"MM"
3030 RETURN
3040 :
3500 REM **** FIND ****
3510 DR=FW:DS=5:GOSUB7000:REM MOVE
3520 IF(PEEK(DATREG)AND192)=NB THEN 3510
3530 RETURN
3540 :
4000 REM **** TEST BUMPERS ****
4010 IF(PEEK(DATREG)AND192)=RB THEN SS=RT:
     GOSUB5000:RETURN
4020 IF(PEEK(DATREG)AND192)=LB THEN SS=LF:
     GOSUB5000:RETURN
4030 RETURN
4040 :
5000 REM **** SEARCH (SS) ****
5010 DR=FW:DS=8:GOSUB7000:REM MOVE
5020 IF(PEEK(DATREG)AND192)=NB THEN 5010
5030 RETURN
5040 :
6000 REM **** PROBE (WY,SP) ****
6010 IF WY=RT THEN OW=LF
6020 IF WY=LF THEN OW=RT
6030 DR=BW:DS=30:GOSUB7000:REM MOVE
6040 DR=WY:AG=90:GOSUB7500:REM TURN
6050 DR=FW:DS=SP:GOSUB7000:REM MOVE
6060 DR=OW:AG=90:GOSUB7500:REM TURN
6070 DR=FW:DS=8:GOSUB7000:REM MOVE
6080 IF(PEEK(DATREG)AND192)=NB THEN 6070
6090 RETURN
6100 :
7000 REM **** MOVE (DR,DS) ****
7010 POKE DATREG,(PEEK(DATREG)AND 1)OR DR
7020 PL=PD*DS
7030 FOR I=1 TO PL:GOSUB8000:NEXT I
7040 RETURN
7050 :
7500 REM **** TURN (DR,AG) ****
7510 POKE DATREG,(PEEK(DATREG)AND 1)OR DR
7520 PL=PA*AG
7530 FOR I=1 TO PL:GOSUB8000:NEXT I
7540 RETURN
7550 :
8000 REM **** PULSE ****
8010 POKE DATREG,PEEK(DATREG)OR 8
8020 POKE DATREG,PEEK(DATREG)AND 247
8030 RETURN
```

ON EDGE

Our series of articles describing vertical software continues with a look at a suite of programs based on assessments of personality characteristics and designed to help business people plan strategies for working with colleagues and clients.

An intermediate step toward artificial intelligence has been taken by Human Edge Software, a California-based programming house. Human Edge programs are sophisticated commercial decision-making tools that work quickly through large amounts of data supplied by the user, evaluate the data according to stored criteria, then produce a recommended course of action. The Human Edge is a suite of four programs — Communications Edge, Sales Edge, Management Edge and Negotiation Edge — for IBM and IBM-compatible machines. The complete suite costs nearly £1,000 and claims to 'increase a user's individual professional skills' in the areas specified in the names of the individual programs. A scaled-down version, called Mind Prober, is available for the Commodore 64, Apple II and Macintosh, using the same techniques, but we will focus here on the four-program suite.

The programs are said to be the outcome of more than ten years of development, involving the work of behavioural scientists and business experts and incorporating new techniques such as human factors analysis, expert systems technology and decision theory mathematics.

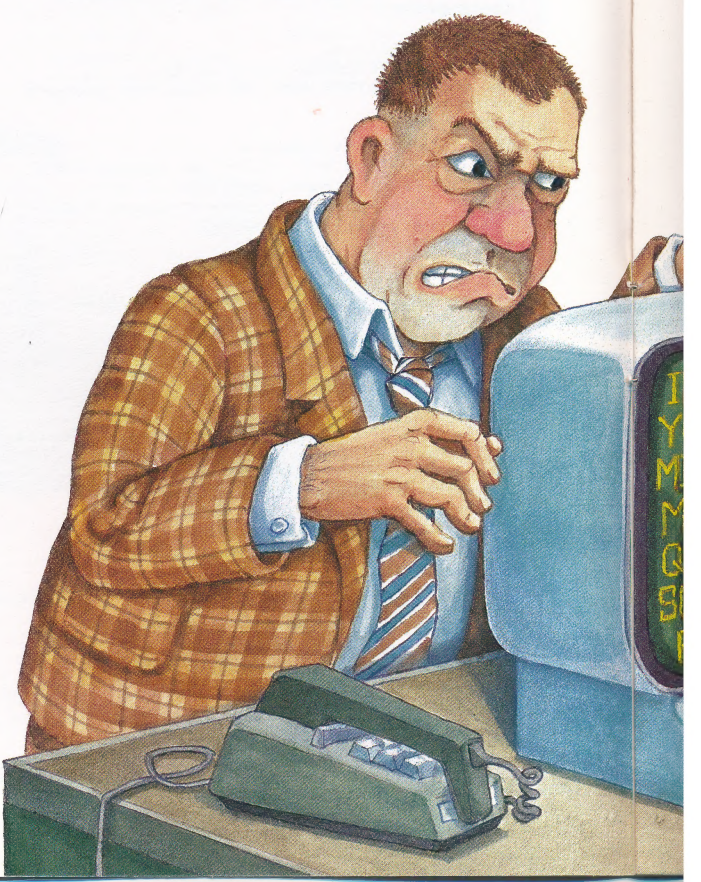
This description sounds rather dramatic and coupled with the cost of the programs could intimidate the potential user. However, the programs are easy to use and can be operated fully after less than an hour's self-teaching. They are all menu-driven, being built around lengthy questionnaires consisting of a series of carefully worded statements with which users are asked to agree or disagree. The statements query the significant personality characteristics of the user, as well as his sales prospects, current clients, company subordinates and superiors, and whichever aspect of business relations the user wishes to investigate.

Responses are evaluated by the program and a detailed report, including a recommended course of action, is prepared. The recommendation may be a suggested opening approach to a new client, an effective closing strategy for a difficult sales prospect, or negotiation techniques to use with employees or employers.

Each program begins with a self-assessment questionnaire, presenting statements such as: 'I take charge in most meetings', 'I argue with others more than most', 'I am somewhat impulsive', and so on. The user decides whether the statement is an accurate description of himself, then enters his response. The self-assessment tool has been prepared effectively, with a considerable overlapping of questions as an internal measure of validity. Thus, the user's responses to 'I am somewhat impulsive' and the later statement 'I sometimes act without thinking' will be evaluated against one another for consistency. When this section is complete, the responses are stored on disk. They can be updated and reused at any time.

Once the self-assessment has been made, the user is asked to agree or disagree with a series of adjectives as they relate to the object of enquiry. Words such as talkative, apprehensive, independent, achieving, ambitious, courteous, flaunting and empathetic are used to help the user gauge his client, employer or subordinate. When working through this list, it might be helpful to have a good American dictionary at hand to help in understanding some of the terms, since the Oxford English Dictionary, doesn't recognise 'empathetic'. A common definition would be 'aware of another, compassionate', although the more common English usage is 'empathic'.

The user can move through the list of adjectives and change responses at any time. As with the self-



DAVID HIGHAM

assessment, the list is saved to disk, though it can also be updated in the light of further experience.

The self-assessment need be completed only once, then it can be recalled from disk and used to relate to any of several 'counterpart' lists. Up to eight counterpart assessments can be saved on disk. When two completed lists are present, the program takes the responses and evaluates them, then prepares a report that synthesises the characteristics of the user and his subject.

The following report, generated by Communication Edge, is based on a real user and counterpart (the user's teenage son), the latter being described as Mr.T (for Test). The report is presented as though the computer were speaking directly to the user:

'Your flexible, stable approach to people will be needed in communicating with Mr. T. He is a very private person who prefers being alone and who has very little patience with small talk or socialising. Expect a cynical or suspicious attitude as you solicit his ideas and feelings. In speaking to him, make few assumptions that he understands you. Be clear, concise and direct.'

'In contrast to your even-tempered style, Mr. T angers quickly and can even appear angry before the conversation begins. He may try to force his opinions on you. Remain cordial, despite this approach. Also be prepared for his unpredictablility. He can speak impulsively one minute, while choosing every word with caution the next. Take on the role of guiding the process of the meeting. Paraphrase his comments to achieve clarity and agreement.'

Parents will probably recognise the description of a typical adolescent, although Communication Edge asks no questions about the age of the subject (the person's sex is considered, however).

The vocabulary used by the reports will be familiar to those who read newspaper advice columns or who participate in similar personality quizzes. The user is generally portrayed in sympathetic terms ('even tempered, flexible, stable'), while the subject is less well-favoured ('angers quickly, unpredictable, cynical, suspicious'). This is probably designed to reinforce the user's self-image, and perhaps to fit current American management thinking of business as warfare, with the sales prospect or customer as the enemy whose resistance must be overcome.

This becomes even more obvious in Negotiation Edge, which recommends such strategies (printed in headline capitals) as:

USE MR T'S KNOWLEDGE TO YOUR ADVANTAGE
SET MR T UP WITH EARLY CONCESSIONS
GET MR T IN THE HABIT OF SAYING "YES"
VEIL YOUR THREATS
EXAGGERATE YOUR PROBLEMS
DOWNPLAY YOUR PROFIT
SETTLE ON A GOOD NOTE

It seems to be assumed by the developers that a user will have only one of the four programs, since each requires the user to complete a separate self-assessment, presenting more or less identical questions in a slightly different order. In view of the price, this may be a reasonable assumption, but since a major organisation would be most likely to use them as management tools it would have been helpful to be able to use the same self-assessment in all four. But as the programs are so similar one could simply take the least expensive of the suite, Communication Edge, and adapt its reports to fit a variety of situations.

A serious weakness in the programs, if they are to be used as management tools, is their inability to learn from experience, leaving aside the user's own updating of assessments. For example, it would be valuable for the user to be able to enter the results of a proposed strategy so that it could be modified in the light of experience, especially in evaluating subjects of whom little is known initially. In addition, many of the questions are difficult to answer on a strict 'agree or disagree' basis, and there is no evidence of a tree structure to the questioning that would permit amplification or verification of answers. One solution might be the inclusion of a 'don't know' option, which would then open up the way to a lower-level question that could be answered positively or negatively.

In the final analysis, one either believes in this type of approach to human relations or one does not. Perhaps the most profitable way of using it would be as an aid to careful preparation before an interview, but then the user must beware of taking its advice too literally — at least until computers become real thinking machines.

The Human Edge
A suite of four packages (obtainable separately) for MS-DOS and PC-DOS machines
Prices:
Communication Edge £195.50, Sales Edge £241.50, Management Edge £241.50, Negotiation Edge £287.50
Distributor:
Thorn EMI Computer Software Distributors,
296 Farnborough Road,
Farnborough, Hants
GU14 7NF
Authors
Human Edge Software, California
Format:
Disk

Mind Prober
For the Commodore 64, Apple II and Macintosh
Prices:
Commodore version £17.35, Apple II and Macintosh versions £26
Distributor:
As above
Authors:
As above
Format:
Disk

LANGUAGE CONSTRUCT

Programming languages have several structures built into them for managing often-used operations. One such structure is the keyword, in which a word like PRINT is always used to indicate the process of sending information to the video display. Generally, a keyword is fairly simple and refers to a single action.

By extension, languages must have *constructs*, or built-in routines, for more sophisticated operations, or for sets of operations that accomplish a specified task. An example of this is a loop construct, such as a FOR...NEXT loop. The actual operations to be looped are defined by the programmer, but the ability to execute a loop itself is inherent in the language.

The essentials of any computer language are assignment, decision, addressing and input/output constructs — BASIC, for example, has =, IF...THEN, line numbers, INPUT and PRINT.

LASER PRINTER

A *laser printer* is a high-speed, high-quality device that uses a laser beam to write characters onto an electrically charged light-sensitive surface. The characters are formed from a matrix of many densely-packed dots, and are then beamed onto the photographic surface. This creates a character pattern with an electrical charge. The charged pattern attracts a toner, thus 'developing' the character image. The image is transferred to paper by heat or pressure, and fixed there by passing the paper through a chemical vapour bath. This process is very similar to that used in many office copying machines. Laser printers have full graphics and colour capabilities, and can produce many different typestyles.

LCD

Liquid crystal displays, or *LCDs*, are becoming increasingly popular as video display units for microcomputers. An LCD consists of a seven-segment display, where each segment is filled with a transparent liquid. The display is sandwiched between two electrodes, with a reflective backing material and a clear cover. When current is applied to the electrodes, the liquid in the affected segment becomes opaque, forming a solid bar. The main advantages of LCDs for microcomputers are the small amount of space they need, and their minimal power requirements. These factors make LCDs ideally suited to use in portable computers. Their major disadvantage is lack of speed: a fast typist can type several characters in the time it takes for the LCD to register the first keypress.

LED

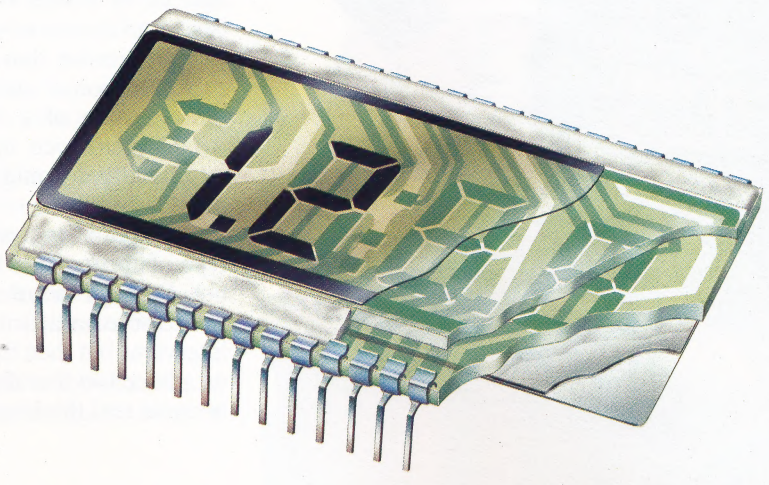
An *LED* is a semiconductor diode that emits light when a current is applied — the initials stand for 'Light Emitting Diode'. Single LEDs, usually red in colour, are often used as warning or informational signals on computer consoles. Like LCDs, LEDs can be combined into a block pattern of seven segments to create alphanumeric characters. This type of display is commonly used on pocket calculators. However, because the power requirements of LEDs are greater than those of a liquid crystal display, LEDs are gradually being phased out in favour of LCDs.

LEXICAL ANALYSER

Because computers have to handle data in certain very specific ways, language compilers must have a special set of routines that take data and alter it to fit the structure needed by the CPU. A *lexical analyser* is such a set of routines; it breaks statements into their component parts — recognised by the analyser as tokens — separating variable names from commands, and so forth. It also adjusts spacing, removing unnecessary spaces and characters, and replacing upper case letters with lower case, or vice versa, as demanded by the compiler.

LIFO

An acronym for 'Last In First Out', *LIFO* refers to the way in which information is stored in a stack. The last item placed in the stack is the first to be removed. The classic analogy relates this process to a stack of plates held in a spring-loaded tray. The top plate is the first used. When it has been removed from the stack, the rest of the plates move up one position. In computer terms, this action is called a *pop*. Adding plates on top presses the rest of the stack down; this is referred to as a *push*. LIFO stacks are commonly used in text buffers. (See also FIFO, page 576.)



A Slice Of Light
The LCD comprises two polarised glass filters sandwiching a very thin layer of liquid crystal. Behind this 'sandwich' is an array of electrodes, which trace out the shapes of the characters. When current is applied to the crystal through the appropriate electrodes, the alignment of the molecules in the overlying liquid crystal changes, making the character shape stand out from the background

TAKING ON BOARDS

Realising that an absence of built-in disk drive interfaces may have detracted from the otherwise highly-regarded Memotech 500 series of micros, the company has recently released the RS128, with full interface facilities. The new machine is a stylish addition to the Memotech range.

Despite being highly regarded machines, the Memotech 500 series of microcomputers — the MTX500 and MTX512 — has been largely overlooked by home computer buyers. Attractive features — such as high resolution graphics, a built-in assembler, a sophisticated BASIC and a unique text-handling language called NODDY — have certainly not detracted from these machines, but their failure to achieve great popular success could be attributed to their falling between two distinct segments of the home computer market.

On the one hand, priced at around £300, the machines are a little expensive for the games player who may think that more sophisticated features are not worth the higher cost. On the other hand, the 'serious' user (Memotech says that the series is aimed at the small business user) may have been deterred by the fact that the 500 series lacks built-in interfaces, which would allow it to be connected to disk drives. These interfaces were available, but they came as separate boards, designed to be fitted to an edge connector inside the machines. This is not altogether surprising coming from a company that made its name by providing add-on boards for the ZX81 (see page 580), but it seems to have failed to impress users who wanted a machine that they could just plug in and run. Memotech seems to have recognised this problem and has introduced the RS128, a machine with interfaces built in.

THE LOOK OF THE MACHINE

At first glance, the RS128 looks identical to the 500 series — it gives the impression of being stylish and a little up-market. Like its siblings, the machine is cased aluminium, instead of the usual plastic, and this makes a Memotech machine considerably heavier than most other micros. There is a standard QWERTY keyboard and a numeric keypad, which holds some of the commands for the NODDY text programming language. There are also eight programmable function keys to the right of the keypad. The keys have an excellent feel and are built to a high professional standard.

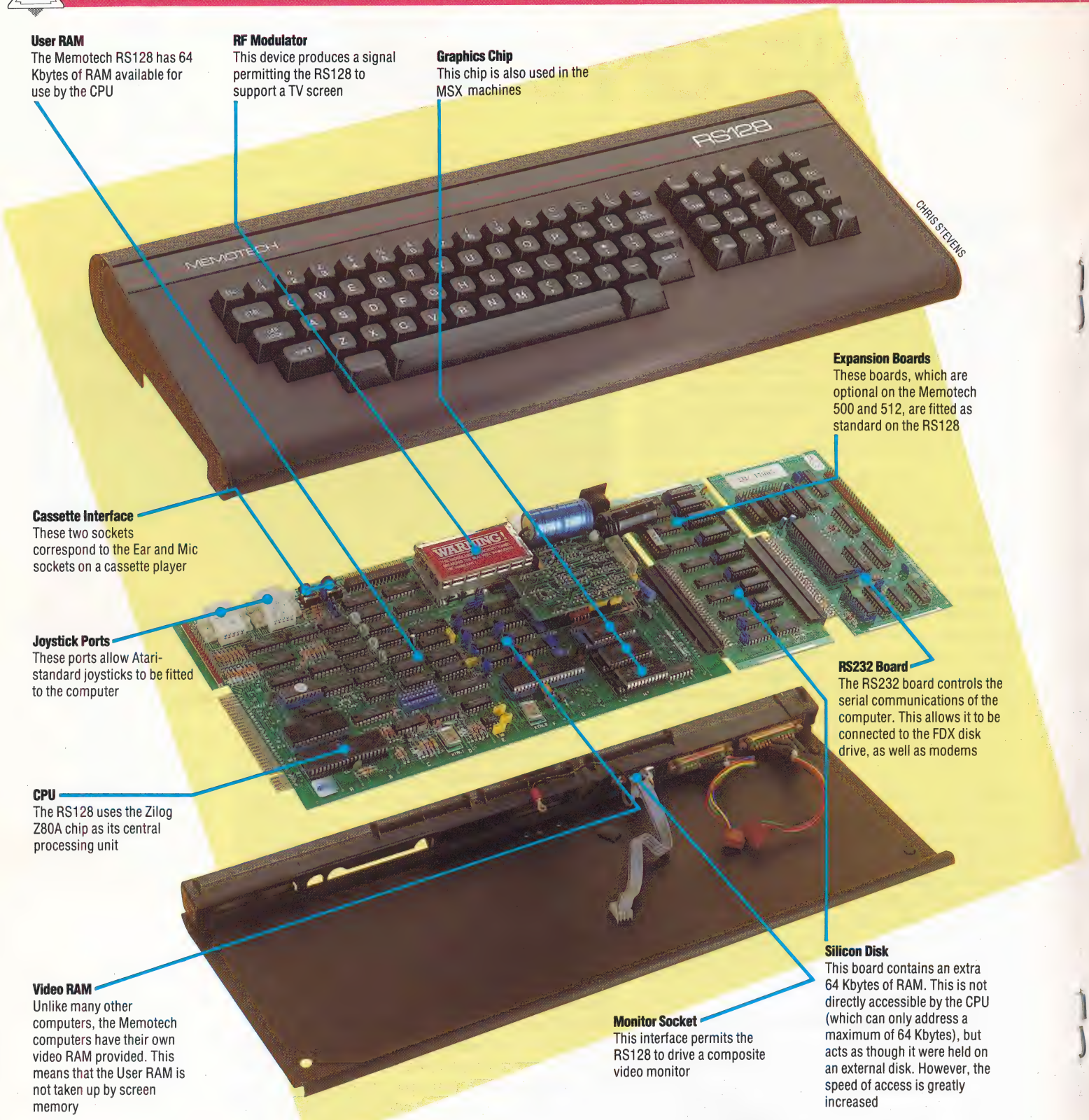
There are a few minor niggles with the layout: the Return key is not much larger than the ordinary keys and touch typists may at first have difficulty in locating it, and the Delete key is not on the typewriter keyboard itself, but located on the numeric keypad instead. There is a Backspace key in the top right hand corner, but unlike most computer keyboards, where the Backspace also acts as a Delete-left key (known as a 'destructive backspace'), on the Memotech it is simply a cursor-left.

CHRIS STEVENS

Much Improved
The Memotech RS128 is an improved version of the MTX500 series. This new model is fitted with twin RS232 sockets, which enable the machine to run the FDX floppy disk drives. This means that the computer is especially attractive to the serious home micro user or the small business user

On the back of the machine there is a number of interfaces. Some of these were provided with the 500 series, and others are recent additions. On the far left of the machine is a pair of RS232 ports, which enable the machine to be connected to FDX floppy disk drives. These ports can also be used for other purposes, such as serial printers and networked communications. To the right of the RS232s is a composite video jack and a hi-fi jack — the latter allows the computer's sound to be amplified through a normal stereo system. The power socket and RF jack come next, followed by a Centronics-type printer interface. The cassette interface consists of a pair of microjack sockets, for EAR and MIC, in the same style as the Sinclair Spectrum. Finally, there is a pair of nine-pin Atari-style joystick ports.

The interface ports are labelled in white lettering, which can be clearly read from the back of the machine. This would seem to allow peripherals to be plugged in without having to lean over to look at the back. Unfortunately, Memotech has set the ports into depressions in the



machine so that the user still has to peer over to locate the plugs.

The 24 by 40 character BASIC screen is divided into three sections, which are best shown on power-up. The top 19 rows are the main screen, where program listings are scrolled. Below this is the EDIT screen, where new lines are entered. At the bottom of the screen is a single line for displaying error messages. Like the Sinclair machines, program lines are altered by use of an EDIT command. Furthermore, the operating system will not allow a line to be inserted into the program from the EDIT screen if the line contains a syntax error.

The BASIC itself is a close relative of MSX BASIC containing such commands as SOUND, PAPER, INK, and CIRCLE. However, the BASIC also contains some useful commands not available in MSX BASIC. These commands, on the whole, relate to the screen-handling capabilities of the machine.

As an example, the command CSR x,y will position the cursor at the point with the co-ordinates (x,y) on the screen. A more powerful command is CRVS, which enables the user to define a window anywhere on the screen. Text or graphics can be displayed within these windows.

There are also commands built into the language to enable the control of sprites. A particularly useful command in this respect, GENPAT, allows you to set the sprite pattern, rather than having to put the pattern into data statements. The Memotech graphics are provided by the TMS9929A Video chip — which is the one specified for MSX machines.

The central processor of the Memotech machines is the Z80, and this of course, enables them to run the CP/M operating system. Many small computer manufacturers choose the Z80 processor because it runs CP/M, which avoids the problem of having to generate a large software base before the customers can take full advantage of a new computer. Of course, to really make the most of CP/M, the computer has to have an 80-column screen, and, unusually, Memotech has provided an 80-column card inside the disk drive. The drives themselves are double-sided and double-density, and have a transfer rate to the computer of 9,200 baud.

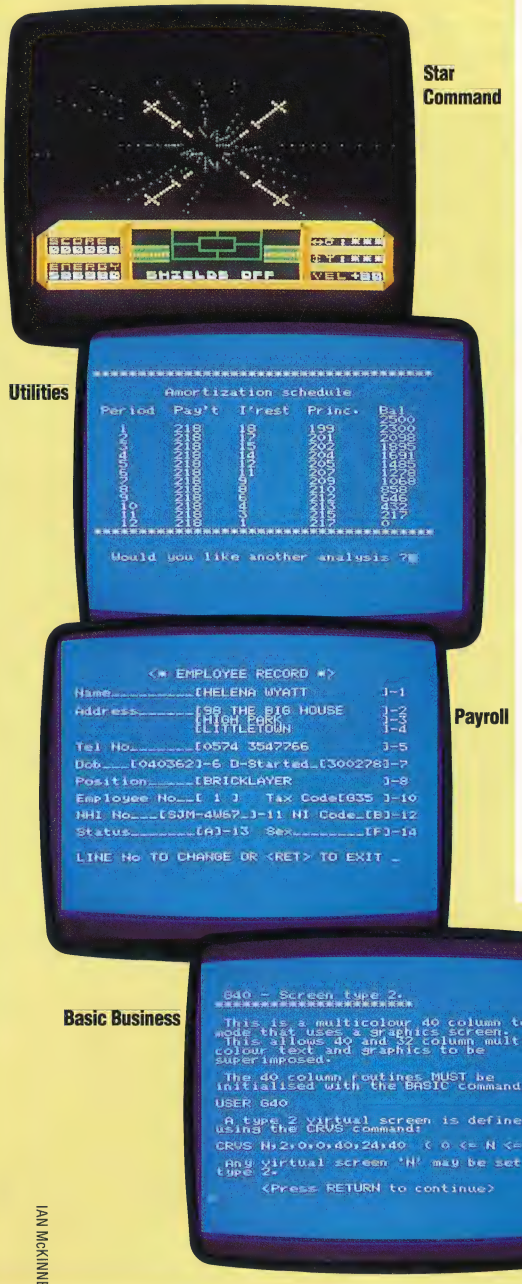
Provided with the disk drive is a package of bundled software. Apart from a CP/M systems disk, the package includes the NewWord word processor, the SuperCalc spreadsheet, Compact and Televideo — which allow the drives to read disks written in other disk formats (Memotech claims this includes IBM disks) — and Contact, which enables the second RS232 port to link into a networked system.

The RS128 has 128 Kbytes of RAM on board. However, as it uses an eight-bit processor, and is only able to address 64 Kbytes, the other 64 Kbytes are provided as a 'silicon disk'. A silicon disk stores files and programs in exactly the same way as a floppy disk, but as it is held on chips it is up to 50 times faster than a conventional floppy disk. Information held on a silicon disk is transferred to addressable RAM when it is required. At the end of a work session, the data can be permanently stored on a floppy disk.

The manual provided with the machine is much larger than those usually provided with home computers, although this is predominantly because it has not been typeset, and there is not a great deal more information included. However, Memotech has included all the technical details a user might need, including circuit diagrams, pinouts and operating system calls.

In upgrading the Memotech 500 series to the RS128, the company has made strenuous efforts to produce a business standard machine. At £399, the machine certainly looks worthy competition for the Sinclair QL, the BBC Micro and the Commodore Plus/4. However, it remains to be seen whether the machine will generate sales to compare with those of its rivals.

FDX Twin Disk Drive
The FDX twin disk drive allows the computer to run the popular CP/M operating system. Each of the 5¼ inch disk drives can store up to 500 Kbytes of information



IAN McKINNELL

Memotech RS128

PRICE
£399

DIMENSIONS
488×202×56mm

CPU
Z80A running at 4MHz

MEMORY
64K RAM and 24K ROM

SCREEN
40×24 in text mode. Text with graphics mode: 32×24 text and 256×192 pixels in 16 colours. There are also facilities for up to 32 independently controllable sprites

INTERFACES
Cassette (Mic and Ear) ports; I/O interface; two joystick ports; two RS232 ports; hi-fi jack; composite video jack; TV jack; parallel interface

LANGUAGES
BASIC, FORTH, PASCAL

KEYBOARD
57 typewriter keys; keys F and J are recessed for finger location. 12 function keys on a numeric keypad and eight programmable function keys

DOCUMENTATION
The manual provided is the same as that for the MTX500 series. The manual is extremely comprehensive, although perhaps much of the tutorial is pitched a little too high for the beginner

STRENGTHS
The addition of the RS232 boards make the RS128 a very attractive buy for the 'serious' home user and small businessman

WEAKNESSES
Despite the machine's ability to run CP/M, the RS128 is still poorly supported in terms of software written especially for the machine

Software Support

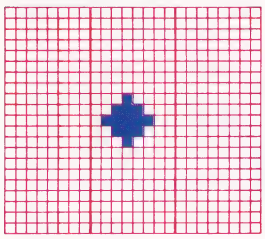
These are some of the business and games packages that are currently available for the Memotech machines. Compared with some other machines, the amount of software available is rather limited, but access to the CP/M pool of software should go some way towards rectifying this problem

SCREEN PLAY

In the last two instalments of this programming project, we designed screen displays for two special locations in the Digitaya adventure, for the Spectrum and BBC Micro. Here, we look at designing and programming these displays on the Commodore 64.

Shoot-Em-Up!
The joystick port's 'shooting' action is accomplished on the 64 through the use of a sprite defined as a projectile. The outlines are set by POKEing zeros into the defining area, then the solid areas are READ from DATA statements and poked into the correct sprite locations

PROJECTILE — SPRITE 1



```
0   16   0
0   56   0
0  124   0
0   56   0
0   16   0
```

Sprite Stretching
The values listed for the joystick port sprite are READ from DATA statements and poked into the appropriate sprite locations. As designed, the joystick port is too compressed, but it can be stretched horizontally (as shown) by changing the value of the horizontal expansion register

The designs of the adventure screens for the BBC Micro and the Spectrum were similar: both computers have high-resolution graphics and easy PRINT formatting in BASIC. The differences lay mainly in the screen dimensions and BASIC dialect words that handle high-resolution plotting. Designing similar screens for the Commodore 64, however, requires a radically different approach. The 64 does have high-resolution facilities, but effectively these are available only to the machine code programmer, as no BASIC commands are provided to handle high-resolution, and performing the relevant PEEKs and POKEs in BASIC to produce high-resolution displays is so slow as to make it unusable in this application. Instead, we must adapt the relatively easy-to-use facilities offered by the Commodore 64.

Graphics characters can be used to build up large letters or other displays by combining different characters. Sprites are a convenient method of introducing high-resolution shapes to the normal Commodore screen. PET characters can be positioned on the screen using either a series of PRINT statements or by POKEing the relevant character code to the screen. We shall demonstrate both methods.

JOYSTICK PORT SCREEN

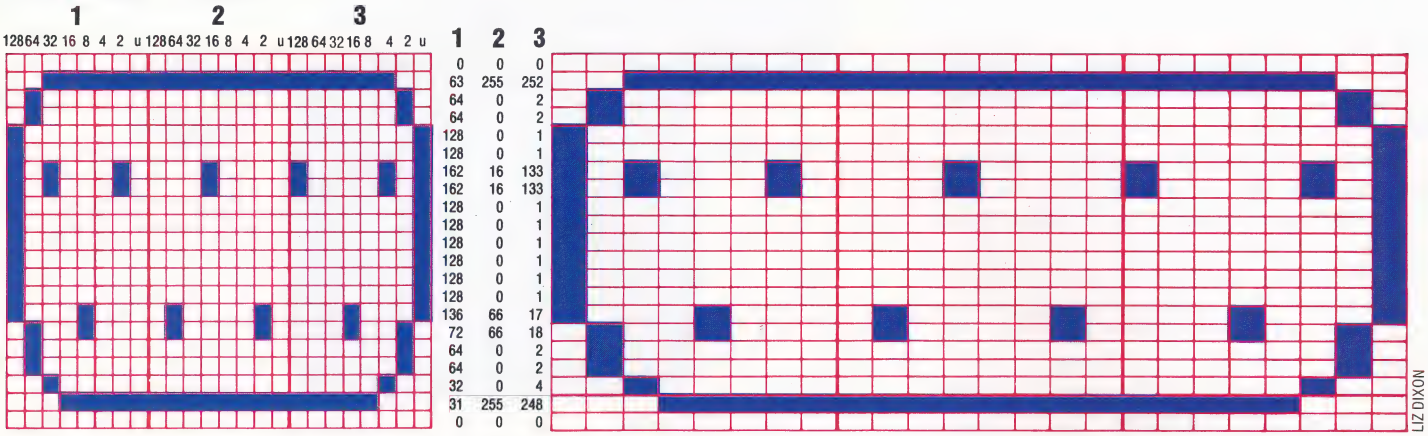
Lines 8020 to 8170 of the Joystick Screen listing are concerned with reading in the data for the two sprites used in this routine. The first, sprite 0, is defined by the first 63 numbers in the group of DATA statements between lines 8450 and 8497, and represents the joystick port (shown in the diagram). Sprite data is usually positioned high in the BASIC program area, but with a large BASIC program this data stands a good chance of being overwritten. An alternative location is the cassette buffer area between locations 832 and 1022, where the data for up to three sprites can be held. This routine stores its sprite data in this safe area.

Sprite 0 is stretched to twice its original width to produce the final displayed shape by setting bit 0 of the horizontal expansion register in line 8170. Notice that all the registers controlling the attributes of sprites, such as colour, position and expansion, are related to the start address for the video control chip (VIC). Remembering that the horizontal expansion register has the address VIC+29 is much easier than memorising its actual memory location (53277). Some sprite attributes require a whole register for each sprite — for example, the X and Y co-ordinate registers — but where the eight bits independently control a function for the eight available sprites, attributes can be controlled by setting and unsetting the appropriate bit in a single register. Sprite 1 is defined by the remaining 13 numbers in the DATA statements and represents an object to be fired out of the joystick port.

Since the 'solid' part of sprite 1 is small (it represents a projectile), it is quicker and easier to enter the 63 bytes of data that define it in two stages. Firstly, POKE 63 zeros into the defining area, and then READ and POKE in the few numbers that define the shape. In this way, we can dispense with the large number of zeros that would otherwise be required as data.

Lines 8190 to 8220 are concerned with the construction of strings constituted from a series of PET graphics characters. LE$ forms a horizontal line the width of the screen by combining 40 of the special PET characters on the front right of the C key. DW$ is a series of cursor-down characters. LS$ and RS$ are groups of left and right diagonals (on the front right of the N and M keys) that are used to form a herringbone pattern in the foreground. This pattern introduces depth and perspective to the scene.

JOYSTICK PORT — SPRITE 0



1	2	3
0	0	0
63	255	252
64	0	2
64	0	2
128	0	1
128	0	1
162	16	133
162	16	133
128	0	1
128	0	1
128	0	1
128	0	1
128	0	1
128	0	1
136	66	17
72	66	18
64	0	2
64	0	2
32	0	4
31	255	248
0	0	0

LIZ DIXON

The 'Shoot' routine at line 8310 chooses a random point at the bottom of the screen and directs sprite 1 down to it, the process repeating until the player presses a key. The screen colours are reset to normal, the screen is cleared and the sprites are turned off before returning to the main program. To use this subroutine with Digitaya, the following line should be inserted:

```
3845 GOSUB 8000:REM JOYSTICK PORT PICTURE
```

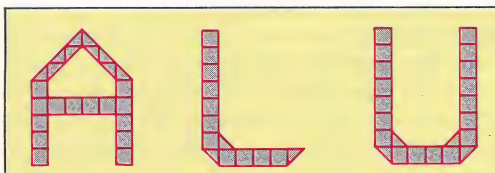
The other listing provides a graphics display for the ALU location in Digitaya and demonstrates different methods of displaying characters on the screen. Lines 7040 to 7090 read a number of DATA statements and POKE the values straight into the screen area. The corresponding location in the colour area is also POKEd with the colour code for the character. In this example the colour code is 2, causing characters to be displayed in red.

A rather unusual trick is used to cause the large letters ALU to scroll down the screen. The first line of graphics character codes that go to make up the letters ALU are POKEd to the second line on the screen. The subroutine at line 7680 is then called, causing the screen to scroll down one line. The second line of codes can then be POKEd into the same screen area as the first, and the subroutine is called again. Repeating this for each of the eight lines of code makes the letters ALU appear to scroll from the top of the screen.

Two other methods of presenting character data to the screen are demonstrated. Characters can be PRINTed directly, as at lines 7130 and 7140, or read as a data string to be PRINTed, as is the case with the question mark design at lines 7170 and 7590 to 7670. This second method allows ease of design within the DATA statements.

To use this routine add the following line:

```
4565 GOSUB 7000: ALU PICTURE
```



LIZ DIXON

Letter Writing
The ALU location for Digitaya is created from three PET low-resolution graphics characters, as shown. The large letters formed appear to scroll down from the top of the screen into their resting position

ALU Screen

```
7000 REM **** ALU PICTURE S/R ****
7010 VIC=53248:CS=55296:SC=1024
7020 PRINT CHR$(147):REM CLEAR SCREEN
7030 POKE VIC+32,0:POKE VIC+33,0:REM SET SCREEN/BORD
7040 CC=0
7050 FOR J=1 TO 8:GOSUB7680:REM SCROLL
7060 FOR I=47 TO 72
7070 READ A:CC=CC+A:POKE SC+I,A:POKE CS+I,2
7080 NEXT I,J
7090 READ CS:IF CS<>CC THEN PRINT"CHECKSUM ERROR":STOP
7100 GOSUB7680:REM SCROLL
7110 PRINTCHR$(158):REM TEXT YELLOW
7120 FOR I=1 TO 8:PRINT:NEXT I:REM MOVE DOWN
7130 PRINTTAB(9)"AND";SPC(7);"OR";SPC(8);"NOT"
7140 PRINTTAB(10)"O";SPC(9);"O";SPC(9);"O"
7150 PRINTCHR$(28):REM TEST RED
7160 REM ** QUESTION MARK **
7170 FOR I=1 TO 9:READ Q$:PRINTTAB(16)Q$:NEXT I
7180 REM **** AWAIT KEY AND RESET ****
7190 GET A$:IF A$="" THEN 7190
7200 POKE VIC+32,14:POKE VIC+33,6:REM SCREEN/BORD
7210 PRINTCHR$(154):REM LT BLUE TEXT
7220 PRINTCHR$(147):REM CLEAR SCREEN
7230 RETURN
7240 REM **** SCREEN DATA ****
7250 REM ** ROW 1 **
7260 DATA 160,32,32,32,32,160,32,32,32,32
7270 DATA 95,160,160,160,160,160,105
7280 DATA 32,32,32,32,95,160,160,160,160,105
7290 REM **ROW 2 **
7300 DATA 160,32,32,32,32,160,32,32,32,32
7310 DATA 160,223,32,32,32,32,32,32,32,32
7320 DATA 160,223,32,32,233,160
7330 REM ** ROW 3 **
7340 DATA 160,32,32,32,32,160,32,32,32,32
7350 DATA 160,32,32,32,32,32,32,32,32,32
7360 DATA 160,32,32,32,32,160
7370 REM ** ROW 4 **
7380 DATA 160,160,160,160,160,160,32,32,32,32
7390 DATA 160,32,32,32,32,32,32,32,32,32
7400 DATA 160,32,32,32,32,160
7410 REM ** ROW 5 **
7420 DATA 160,32,32,32,32,160,32,32,32,32
7430 DATA 160,32,32,32,32,32,32,32,32,32
7440 DATA 160,32,32,32,32,160
7450 REM ** ROW 6 **
7460 DATA 233,105,32,32,95,223,32,32,32,32
7470 DATA 160,32,32,32,32,32,32,32,32,32
7480 DATA 160,32,32,32,32,160
7490 REM ** ROW 7 **
7500 DATA 32,233,105,95,223,32,32,32,32,32
7510 DATA 160,32,32,32,32,32,32,32,32,32
7520 DATA 160,32,32,32,32,160
7530 REM ** ROW 8 **
7540 DATA 32,32,233,223,32,32,32,32,32,32
7550 DATA 160,32,32,32,32,32,32,32,32,32
7560 DATA 160,32,32,32,32,160
7570 DATA 14463:REM CHECKSUM
7580 REM ** QUESTION MARK DATA **
7590 DATA "  ?????  "
7600 DATA "?      ?"
7610 DATA "?      ?"
7620 DATA "     ??  "
7630 DATA "    ?    "
7640 DATA "    ?    "
7650 DATA "    ?    "
7660 DATA "         "
7670 DATA "    ●    "
7680 REM **** SCROLL SCREEN S/R ****
7690 POKE 218,160
7700 PRINTCHR$(19);CHR$(17);CHR$(157);CHR$(148)
7710 RETURN
```

Joystick Port Screen

```
8000 REM **** JOYSTICK PICTURE S/R ****
8010 PRINTCHR$(147):REM CLEAR SCREEN
8020 S0=832:S1=S0+64:REM SPR DATA START ADDR
8030 CC=0:VIC=53248:REM START OF VIC CHIP
8040 FOR I=S0 TO S0+62:READ A:CC=CC+A:POKE I,A:NEXT
8050 FOR I=S1 TO S1+62:POKE I,0:NEXT I
8060 FOR I=S1+25 TO S1+37:READ A:CC=CC+A:POKE I,A:NEXT I
8065 READ CS:IF CS<>CC THEN PRINT"CHECKSUM ERROR":STOP
8070 POKEVIC+33,0:POKEVIC+32,0:REM SET SCREEN/BORDER
8080 REM ** SET SPRITE POINTERS **
8090 POKE 2040,S0/64:POKE 2041,S1/64
8100 REM ** SET VIC SPRITE CONTROL REGS **
8110 POKE VIC+39,7:REM SET SPR 0 COLOUR
8120 POKE VIC+40,7:REM SET SPR 1 COLOUR
8130 POKE VIC,65:REM SET SPR 0 X COORD
8140 POKE VIC+1,70:REM SET SPR 0 Y COORD
8150 POKE VIC+2,74:REM SET SPR 1 X COORD
8160 POKE VIC+3,70:REM SET SPR 1 Y COORD
8170 POKE VIC+29,1:REM EXPAND SPR 0 HORIZ
8190 LE$="":FOR I=1 TO 40:LE$=LE$+CHR$(195):NEXT I
8200 DW$="":FOR I=1 TO 25:DW$=DW$+CHR$(17):NEXT I
8210 LS$="":FOR I=1 TO 11:LS$=LS$+CHR$(206)+" ":NEXT I
8220 RS$="":FOR I=1 TO 11:RS$=RS$+CHR$(205)+" ":NEXT I
8230 PRINTCHR$(158):REM TEXT YELLOW
8240 PRINTCHR$(19):PRINT TAB(2)"JOYSTICK PORT"
8250 PRINTCHR$(154):REM TEXT LT BLUE
8260 PRINTCHR$(19);LEFT$(DW$,17);LE$
8270 PRINT CHR$(145);
8280 PT$=LEFT$(LS$,19)+LEFT$(RS$,21)
8290 PT$=PT$+RIGHT$(LS$,19)+RIGHT$(RS$,21)
8300 FOR I=1 TO 3:PRINT PT$;:NEXT I
8305 POKE VIC+21,3:REM TURN ON SPR 0 & 1
8310 REM ** SHOOT **
8320 YB=240:Y1=70
8330 X1=74:X=INT(RND(1)*150)+24
8340 G=2*(X-X1)/(YB-Y1)
8350 FOR Y=Y1 TO YB STEP 2
8360 X1=X1+G:POKE VIC+2,X1:POKE VIC+3,Y
8370 NEXT Y
8380 GET A$:IF A$="" THEN 8330
8390 REM ** RESET SCREEN **
8400 POKE VIC+21,0:REM TURN SPRITES OFF
8410 POKE VIC+32,14:POKEVIC+33,6:REM RESET SCREEN/BORD
8420 PRINT CHR$(147):REM CLEAR SCREEN
8430 RETURN
8440 REM **** SPRITE DATA ****
8450 DATA 0,0,0,63,255,252,64,0,2,64,0,2
8460 DATA 128,0,1,128,0,1,162,16,133,162,16,133
8470 DATA 128,0,1,128,0,1,128,0,1,128,0,1
8480 DATA 128,0,1,128,0,1,136,66,17,72,66,18
8490 DATA 64,0,2,64,0,2,32,0,4,31,255,248
8495 DATA 0,0,0,16,0,0,56,0,0,124,0,0,56,0,0,16
8497 DATA 3701:REM CHECKSUM
```

!

LOGO is an ideal language for exploring mathematics. You begin by developing a few procedures for basic arithmetic tasks, and then use these primitives to perform quite complex calculations. We demonstrate how the language is used to calculate factorials, and show how a few results are transformed into 'factorial trees'.

In how many ways can you arrange four people in four chairs around a table? The first person can be seated in any one of the four seats, but once he has sat down only three choices remain for the second, then there are two choices for the third, and for the last there is only one place left. So, the total number of different arrangements is $4 \times 3 \times 2 \times 1$. This is usually written as 4! and read as '4 factorial'. Factorials are often found in mathematics problems concerning arrangements, combinations and probabilities.

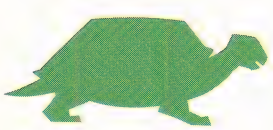
It is simple to write a recursive definition to calculate factorials. First of all, we must note that the factorial of 0 is defined as 1. The factorial of any non-zero positive number — x say — is the factorial of x−1 multiplied by x. Translating this into a program we get:

```
TO FACTORIAL :X
  IF :X = 0 THEN OUTPUT 1
  OUTPUT (FACTORIAL :X — 1) * :X
END
```

To try it out, type PRINT FACTORIAL 6 — the result should be 720.

This procedure works fine up to 12, but beyond this the numbers become too large to be held as integers by the computer. On the Commodore 64, for example, PRINT FACTORIAL 13 gave 6.22702E9 — that is, 6.22702 times 10^9. This is hardly satisfactory, as the last four digits have been lost. There are many reasons (including simple curiosity) why we might want to know what these remaining digits are. The first thing we need to do, therefore, is extend the arithmetic capabilities of LOGO so that it can calculate to greater than seven figure accuracy.

To simplify matters, we will only consider positive integers. We'll represent the integers as lists — so we will represent 1,234,567 as [1 2 3 4 5 6 7]. The following two procedures will do addition on such numbers. Try them out with PRINT LONGADD [1 2 3] [5 6 9] — the result should be [6 9 2]:



```
TO LONGADD :X :Y
  OUTPUT LONGADD1 :X :Y 0
END

TO LONGADD1 :X :Y :CARRY
  IF (ALLOF (EMPTY?:X) (EMPTY? :Y) (:CARRY =
  0)) THEN OUTPUT []
  TEST EMPTY? :Y
  IFTRUE IF :CARRY = 0 THEN OUTPUT :X ELSE
  OUTPUT LONGADD1 :X [1] 0
  TEST EMPTY? :X
  IFTRUE IF :CARRY = 0 THEN OUTPUT :Y ELSE
  OUTPUT LONGADD1 [1] :Y 0
  MAKE "SUM (LAST :X) + (LAST :Y) + :CARRY
  OUTPUT LPUT REMAINDER :SUM 10 LONGADD1
  BUTLAST :X BUTLAST :Y QUOTIENT :SUM 10
END
```

These procedures work in much the same way as we would do additions on paper, adding from the left and incorporating any number carried from the previous column.

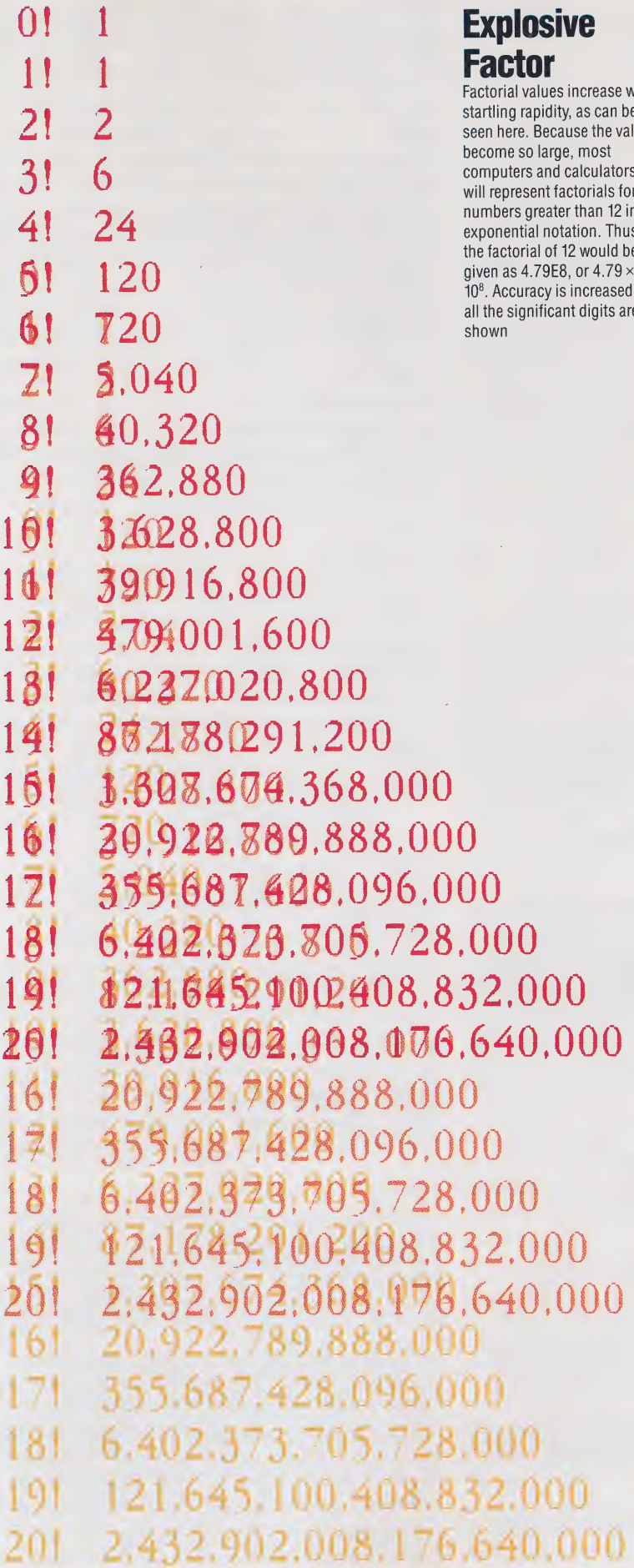
Subtraction is a similar process. However, we have included a routine to delete leading zeros from an answer, so that we don't end up with results such as [0 0 0 7 8].

```
TO LONGSUB :X :Y
  OUTPUT STRIPZEROS LONGSUB1 :X :Y 0
END

TO LONGSUB1 :X :Y :BORROW
  IF (ALLOF (EMPTY? :X) (EMPTY?:Y) (:BORROW =
  0)) THEN OUTPUT [0]
  TEST EMPTY? :Y
  IFTRUE IF :BORROW = 0 THEN OUTPUT :X ELSE
  OUTPUT LONGSUB1 :X [1] 0
  IF EMPTY? :X THEN PRINT [SORRY, I CAN'T
  HANDLE A NEGATIVE RESULT] TOPLEVEL
  MAKE "DIFF (LAST :X) — (LAST :Y) — :BORROW
  IF :DIFF <0 THEN OUTPUT LPUT (10 + :DIFF)
  LONGSUB1 BUTLAST :X BUTLAST:Y 1
  OUTPUT LPUT :DIFF LONGSUB1 BUTLAST :X
  BUTLAST :Y 0
END

TO STRIPZEROS :X
  IF EMPTY? :X THEN OUTPUT [0]
  IF NOT ((FIRST :X) = 0) THEN OUTPUT :X
  OUTPUT STRIPZEROS BUTFIRST :X
END
```

Long multiplication is slightly more complicated. We'll implement it using the technique normally taught in schools. For example, supposing we want to multiply 123 by 338. The problem is split up into three parts: first we multiply 123 by 8; then we multiply 123 by 330; and, finally, we add the two results together. This method depends on the fact that the second stage can be broken down into two sub-stages: firstly, 123 is multiplied by 33; and then a zero is placed at the end of the result. To multiply a number by 33 clearly involves the use of



IAN McKINNELL

Explosive Factor

Factorial values increase with startling rapidity, as can be seen here. Because the values become so large, most computers and calculators will represent factorials for numbers greater than 12 in exponential notation. Thus, the factorial of 12 would be given as 4.79E8, or 4.79×10^8. Accuracy is increased if all the significant digits are shown

recursion. The procedure LONGMULT controls this general strategy:

```
TO LONGMULT :X :Y
   IF EMPTY? BUTLAST :Y THEN OUTPUT
   LONGMULT1 :X LAST :Y 0
   OUTPUT LONGADD (LONGMULT1 :X (LAST :Y)0)
   (LPUT "0 LONGMULT :X BUTLAST :Y)
END
```

The details of multiplying a line by a single digit are carried out by LONGMULT1:

```
TO LONGMULT1 :X :NO :CARRY
      TEST EMPTY? :X
      IFTRUE IF :CARRY = 0 THEN OUTPUT [] ELSE
      OUTPUT (LIST :CARRY)
      MAKE "PROD (LAST :X) * :NO + :CARRY
      OUTPUT LPUT REMAINDER :PROD 10
      LONGMULT1 BUTLAST :X :NO QUOTIENT
      :PROD 10
   END
```

We won't need procedures to perform division for calculating factorials, but you might care to extend the system to cover division for yourself.

We now have a set of primitives for carrying out arithmetic to any degree of precision. The only limitation on the size of numbers that can be handled is the total memory space available to the program.

MAKING MODIFICATIONS

We can now modify our original factorial program to use our new form of long multiplication.

```
TO FACT :X
   IF FIRST :X = 0 THEN OUTPUT [1]
   OUTPUT LONGMULT (FACT LONGSUB :X [1]) :X
END
```

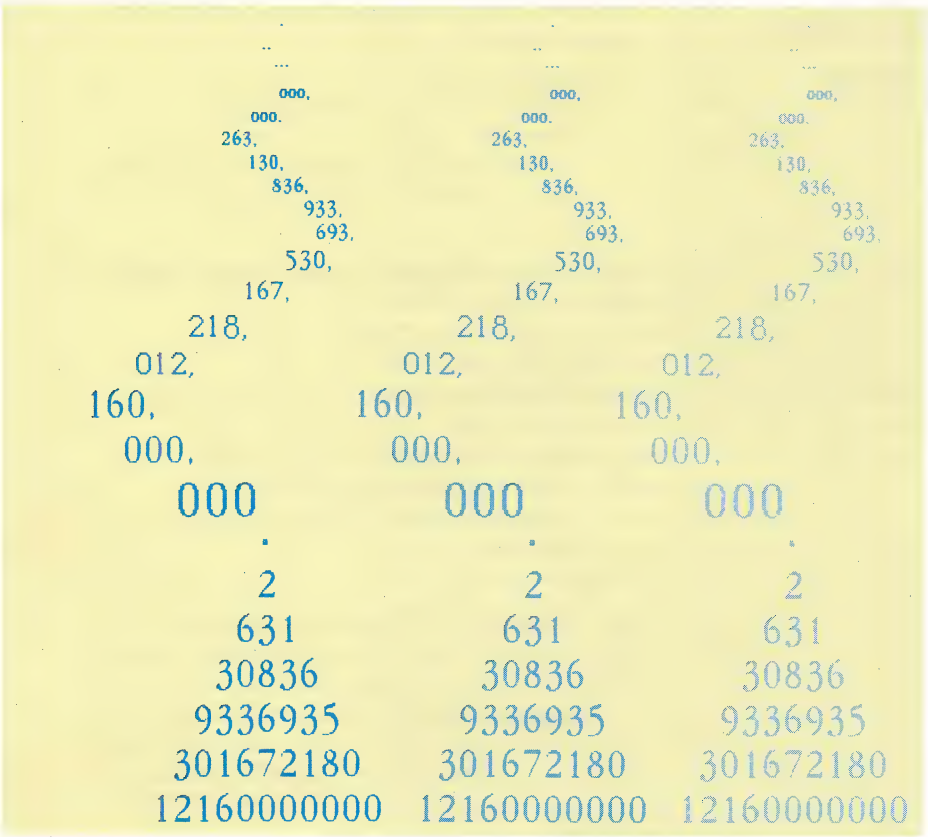
To try it out type FACT [1 3]; you should get [6 2 2 7 0 2 0 8 0 0] as the result. There are problems, however. The calculation process is slow, and — on the Commodore 64 — the largest factorial we obtained before running out of memory was 34!, which has 39 digits (and took some time to be calculated).

The expression of large numbers as lists looks rather unusual, but we can modify the program to overcome this problem by translating back and forth between our usual notation and the list form. We employ two procedures — EXPLODE and IMPLODE — to do this.

EXPLODE 123 outputs [1 2 3] and IMPLODE [1 2 3] outputs 123

```
TO EXPLODE :X
   IF EMPTY? :X THEN OUTPUT[]
   OUTPUT (SENTENCE FIRST :X EXPLODE
   BUTFIRST :X)
END

TO IMPLODE :X
   IF EMPTY? :X THEN OUTPUT"
   OUTPUT (WORD FIRST :X IMPLODE
   BUTFIRST :X)
END
```



Number Tree
Factorial trees are generated by using the leftmost digit of a factorial value as the top of the tree. Subsequent digits are pulled out of the actual value, from left to right, in groups of slightly increasing size. The groups are placed below, and in a symmetrical position to, the cornerstone digit building up a tree shape. This diagram represents the factorial of 32. The value can be read more simply when the numbers are placed in groups of three, as shown

These procedures depend on the fact that numbers in LOGO are treated as words. By using them, we can now define a procedure, F:

```
TO F :X
   PRINT IMPLODE FACT EXPLODE :X
END
```

Which will calculate the factorial of 13 in response to the input: F13.

The result of this calculation — 6227020800 — is a little hard to read as such. It is more usual to insert commas (6,227,020,800), which makes it easier to understand. The following procedures divide the word up into groups of three digits and insert commas.

```
TO ADDCOMMAS :X
   IF ((COUNT :X)<4) THEN OUTPUT :X
   OUTPUT (WORD ADDCOMMAS BUTTHREE :X ",
   LASTTHREE :X)
END

TO BUTTHREE :X
   OUTPUT BUTLAST BUTLAST BUTLAST :X
END

TO LASTTHREE :X
   OUTPUT (WORD (LAST BUTLAST BUTLAST :X)
   (LAST BUTLAST :X) (LAST :X))
END
```

We must also modify F to incorporate these procedures:

```
TO F :X
   PRINT ADDCOMMAS IMPLODE FACT EXPLODE :X
END
```

Using F to print out the first 20 factorials gives some idea of how quickly factorials grow in size (the results are given in the table).

Having obtained the factorials of a range of numbers, we can begin to 'play around' with our results. An American mathematician, for example, once had the brilliant idea of printing out large factorial numbers as trees on the Christmas cards he sent to his friends. Not many factorials have the right number of digits to be printed as trees, but the following procedures will work if it is possible to do so:

```
TO TREE :L
   TREE1 1 :L
END

TO TREE1 :NO :L
   IF EMPTY? :L THEN STOP
   REPEAT ROUND (20 — :NO / 2) [PRINT1 SPACE]
   LINEPRINT :NO :L
   TREE1 :NO + 2 PRUNE :NO :L
END

TO SPACE
   OUTPUT CHAR 32
END

TO LINEPRINT :NO :L
   IF :NO = 0 THEN PRINT "STOP
   PRINT1 FIRST :L
   LINEPRINT :NO — 1 BUTFIRST :L
END

TO PRUNE :NO :L
   IF :NO=0 THEN OUTPUT :L
   OUTPUT PRUNE :NO -1 BUTFIRST :L
END
```

Once again, our controlling procedure must be modified:

```
TO F :X
   TREE IMPLODE FACT EXPLODE :X
END
```

The diagram shows 32! written out as a tree. If you are interested in exploring these factorial trees further, you might like to know that there are only three numbers less than 32 whose factorials can be written as trees. The next larger suitable factorial is 59!

Logo Flavours

For all LCSI versions use:

EMPTYP for EMPTY?
AND for ALLOF
TYPE for PRINT1

There is also a different IF syntax. For example:

IF :CARRY = 0 [OUTPUT []][OUTPUT (LIST :CARRY)]

In place of QUOTIENT :X :Y use DIV :X :Y on the Spectrum and ROUND (:X / :Y) on the Atari.

Use SE for SENTENCE on the Atari.

THE FX EFFECT

Having introduced the BBC Micro's operating system in recent instalments, we return to the subject with a closer examination of OSBYTE calls, a convenient way of accessing many of the OS functions of this computer.

When we first considered how to access the operating system of the BBC Micro (see page 879), we briefly discussed a group of OS calls known as OSBYTE calls. These enable us to modify the behaviour of various parts of the operating system. For example, the OSBYTE call *FX4,1 enables us to change the way in which the computer responds when one of the cursor keys on the BBC keyboard is pressed.

When you realise that there are well over 100 of these calls in the Version 1.2 OS (see page 858), it's not surprising that they offer us a convenient way of accessing many of the OS functions of the BBC Micro. We've already seen that the use of indirect OS calls provides us with insurance against changes in the hardware and software configuration of the machine; OSBYTE provides us with a major method of using OS routines.

Before we go on to examine OSBYTE in detail, it should be pointed out that if your machine has an old Version 0.1 OS, some of the OSBYTE calls mentioned in this course and in the BBC user guide are not actually supported. You can find out which version your machine has by typing *HELP ‹RETURN›.

Let's look first at how OSBYTE is used from BASIC and machine code. Like many BBC OS calls, OSBYTE is vectored (see page 878). The OSBYTE vector is at addresses &20A and &20B. The ways in which we can issue an OSBYTE call are shown below. These will all execute the OSBYTE call mentioned above — *FX4,1:

Using *FX From BASIC	Using USR From BASIC	Using Machine Code
*FX4,1	A%=4:X%=1:D%=USR(&FFF4)	LDA #4 LDX #1 JSR &FFF4

It's clear from these three examples that the address at which we call the OSBYTE routines is &FFF4, and that parameters are passed over to the OSBYTE call in the A, X and Y registers of the 6502 processor. Assigning a value to A%, X% or Y% from BASIC and then calling a machine code routine with the USR or CALL command from BASIC will enter the machine code program called with the A, X and Y registers of the processor holding the values that were in the A%, X% and Y% variables, respectively. The use of USR in the second example enables us to get a result from a machine code program back into a BASIC variable. This is useful with regard to some OSBYTE calls, as they can return information to BASIC — we will discuss this in detail later.

In the third example, we use a short machine code program to make the OSBYTE call; we simply load the necessary registers with the appropriate values and then call OSBYTE at its call address of &FFF4. These examples also show how the parameters (4 and 1) passed to the OS with an FX call, correspond to the parameters passed in the A, X and Y registers when we call OSBYTE routines from machine code or with a USR call.

No matter which method of calling the OSBYTE routine we choose, the contents of the A register always specify which of the many OSBYTE routines is to be used. X and Y registers are then used to pass over parameters to the desired OSBYTE routine. Some OSBYTE calls require no parameters; some require just one parameter to be passed over in the X register; and others, less commonly, require two parameters, passed over in the X and Y registers.

Here are some examples of all these types of OSBYTE calls:

No Parameters	One Parameter	Two Parameters
*FX0	*FX4,1	*FX151,96,200

There are a couple of points to note about calling OSBYTE routines via the use of the *FX call. One is that OSBYTE is totally ignorant of BASIC variables — like all * commands. Executing the code below will give the Bad Command error message:

```
a=4:b=1
*FX a,b
```

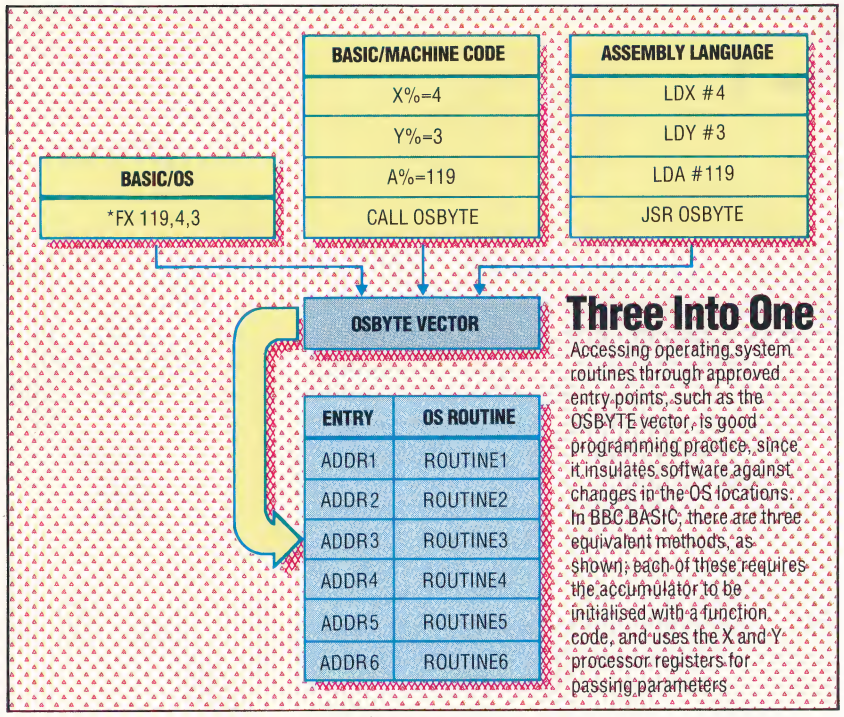
However, we can get round this problem by passing the FX command to the OS using OSCLI:

```
10 DIM C 100
20 a=4
30 b=1
40 $C="*FX "+STR$a+","+STR$b
50 X%=C MOD 256
60 Y%=C DIV 256
70 CALL &FFF7
```

The other point to note about FX calls is that you cannot put anything else on a program line after them. Thus:

```
*FX 4,1:PRINT "Oooops!"
```

will generate another Bad Command — the OS



LIZ DIXON

Three Into One

Accessing operating system routines through approved entry points, such as the OSBYTE vector, is good programming practice, since it insulates software against changes in the OS locations. In BBC BASIC, there are three equivalent methods, as shown; each of these requires the accumulator to be initialised with a function code, and uses the X and Y processor registers for passing parameters.

regards the colon and the PRINT statement as part of the FX command.

Both problems are caused by the fact that the *FX calls are passed through the command line interpreter (CLI) rather than the BASIC interpreter, and the CLI has no 'knowledge' of how to evaluate BASIC variables and deal with multi-statement lines.

As we have shown, it is possible to pass parameters over to OSBYTE in the X and Y registers; it is also possible to read values back from some of the system variables used by the operating system. This can be done by using the USR call or the machine code routine — as we've already shown. You'll probably find the machine code method easier to use when you're interested in getting results back from the OS — the value returned by the USR call has to be decoded to get various bits of information out of it. Parameters are passed back to BASIC in the X and Y registers, and in some of the calls the carry flag is also used to signal error conditions.

The results passed back in this way will obviously depend upon the call — that is, on the value passed to OSBYTE in the A register. Not all OSBYTE calls pass results back to BASIC. However, many of those that do provide us with some useful information about the OS.

Two kinds of information may be passed back by an OSBYTE call. The first is data read from some part of the system, such as the user port, speech processor or system variables. OSBYTE calls passing back this sort of information are referred to as 'read only' calls. A typical example of their use is the OSBYTE call with A=129. This call is used by BASIC to implement the INKEY() function.

The X and Y registers should be set up to pass the required time delay over to the operating system. The X register holds the low byte of the time delay — in centiseconds — and the Y register holds the high byte. Thus, to use this call to wait for up to one second for a keypress, we can use the section of machine code shown below. The X and Y registers pass values back; if the carry flag is set to 0, and the Y register holds a value of 0, then the call was exited by a keypress. The ASCII value of the key thus returned is to be found in the X register. If Y holds 255 and C is set to 1, then no key was pressed in the time period allowed. If C is set to 1 and Y holds 27, this indicates that the Escape key has been pressed.

The following section of code shows how this OSBYTE call can be made. If the carry flag is set on return from the subroutine a branch is made to a further handling routine. This routine may test the value in the Y register to determine whether a key has been pressed or the Escape key has been hit.

```
1000    LDA     #129    /set OSBYTE
1010    LDX     #100    /parameters
1020    LDY     #0
1030    JSR     &FFF4   /make OSBYTE call
1040    BCS     error
1050    RTS
1060    .error          /code to deal with error
```

Other OSBYTE calls, especially those with a value in A of between 166 and 255, are both read and write calls, and they enable us to either read or write certain system variables in the OS. You may begin to wonder how the OSBYTE call knows whether a read or write operation is required; it's actually quite simple.

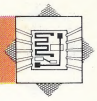
To write a value with the OSBYTE call, the call is made with the X register holding the value we want the OSBYTE call to write and the Y register set to 0. To read a value back from one of these systems variables, X is set to 0 and Y is set to 225. The call is then made. If a value is returned, it resides in the X and Y registers.

THE USES OF OSBYTE CALLS

OSBYTE calls are the 'Civil Servants' of the operating system, being involved in many of the different OS routines. Filing systems, the keyboard, Econet, the Break and Escape keys — all are affected to a greater or lesser extent by OSBYTE calls. The number of different OSBYTE calls available makes it impossible to discuss them all, but here are a few of the more useful ones not covered in detail in the BBC Micro's user guide.

Function keys: *FX18 has no parameters, but is quite useful. It deletes from memory the current function key definitions, so is handy when you want to define a function key more than once in a program.

*FX225 to *FX228: If you've not programmed a function key with a string, you can use these calls to make the red function keys return an ASCII value. For example, *FX225,n will cause function key f0 to return the ASCII code for n when pressed, f1 will return ASCII code (n+1), and so on. *FX226 does the same job for the occasions



when the keys are pressed in conjunction with the Shift key, *FX227 for when the keys are pressed in conjunction with the Control key, and *FX228 works when both the Shift and Control keys are pressed with the function keys.

Video functions and VDU drivers: Most of the control of display in the BBC Micro is performed by writing values to the VDU drivers. However, there are a couple of OSBYTE calls that are helpful in this context.

*FX19: Although this has no parameters, it is useful when you're programming moving graphics. Once executed, it causes the computer to wait until the next frame of the display is drawn before continuing. This results in moving graphics that are less 'flickery'. The call must be made whenever the pause is required. As the display is redrawn 50 times per second, it can be used to generate time delays or to provide interrupts to the CPU.

*FX218: This is a read/write OSBYTE call that informs us of the length of the 'VDU queue'. We have already explained how some VDU codes, when sent through the VDU drivers, expect other codes to follow them (see page 897). The number of bytes that the VDU drivers are still waiting for at any time is the number of bytes in the VDU queue. On the whole, this call is best used to read information back, and the result will be returned in the X register.

*FX20: This call enables us to redefine characters in the ASCII range 32 to 255, rather than the more usual and limited range of user-defined characters. We can thus redefine, in Modes 0 to 6, the characters that are accessed from the normal keyboard. To redefine the characters with ASCII codes in the range 32 to 128 we must set aside memory from our BASIC workspace by setting PAGE to a higher value than usual. The redefinition of these characters is done with the VDU23 call. Characters are redefined in blocks of 32, each block requiring 256 bytes of memory. The only parameter used is passed over in the X register. Full details of this technique are given in the user guide.

Sound: *FX210,n is a useful call for those plagued by noisy 'blast-the-aliens' games. It allows you to disable the sound effects. *FX210,0 enables sound to be heard in the usual fashion, but any other value as the X parameter kills the sound.

*FX211 to *FX214: These calls control the sound generated by CTRL-G or VDU7. They are read/write calls. *FX211,n controls the channel of the sound chip on which the VDU7 sound is generated. *FX12,n controls the volume of the sound generated, or the envelope used to generate the sound. The volume is coded in the same fashion as the amplitude parameter in the BASIC SOUND command — that is, -15 is very loud, 0 is inaudible, and positive numbers are envelope numbers. The value passed over to the X register in the FX call is given by (n-1)*8, where n is the parameter. *FX213,n controls the pitch of the tone generated by VDU7; *FX214,n controls the time for which the tone generated by VDU7 is played.

Escape key: *FX229,n enables the Escape key to be 'turned off'. *FX229,1 disables the Escape key, and *FX229,0 enables normal Escape key action.

*FX220,n enables the user to set up a key to generate the Escape event — n being the ASCII code of the key that you wish to act as Escape. The normal Escape key is disabled. Thus, *FX220,65 causes the Escape event to occur whenever the A key is pressed.

Buffers: *FX21,n flushes — or empties — a buffer. This operation removes any bytes from that buffer. For example, flushing the keyboard buffer removes any keypresses that might have accumulated while a program has been running. If you try a GET command when there are some unprocessed 'keys' in the keyboard buffer, the GET command will fetch a key from the buffer instead of waiting for a keypress to occur. Flushing a sound buffer finishes sound generation on that particular channel, even if there are sound commands waiting to be processed by the sound chip. The buffer to be operated on depends upon the X parameter (see table in the margin).

*FX138,n,m: This inserts the value m into the buffer numbered n. Buffer numbers are the same as for *FX21. Thus, *FX138,0,65 will insert the letter A into the keyboard buffer.

X Parameter	Buffer Operated On
0	Keyboard
3	Printer
4	Sound channel 0
5	Sound channel 1
6	Sound channel 2
7	Sound channel 3

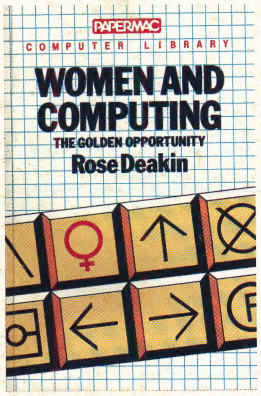
All these OSBYTE calls can, of course, be made by passing the relevant parameters in the A, X and Y registers and making the subroutine call as described earlier.

We have covered all the major functions that come under the control of OSBYTE calls. There are many more that we could examine in detail — you will find them listed in the BBC Micro user guide — but the techniques involved in using them are the same as those described here.

ERRATUM

The User-Defined Commands (USERV) program listing on page 898 was not given in full. Here is the complete listing:

```
5  DIM C 100
10 USERV=&200
20 ?USERV=&00
25 ?(USERV+1)=&0C
30 FOR I%=0 TO 2
   STEP 2
40 P%=&C00
50 [ OPT I%
60    CMP #0
70    BNE notcode
80    TXA
90   .loop
95    JSR &FFE3
100   DEY
110   CPY #0
120   BNE loop
130   RTS
140  .notcode
145   RTS
150 ]: NEXT I%
160 .
200 FOR rep=1 to 10
210 FOR asc=33 TO 48
220 asc$=STR$(asc)
230 rep$=STR$(rep)
240 code$="*CODE
    "+asc$+","+rep$
250 $C=code$
260 X%=C MOD 256
270 Y%=C DIV 256
280 CALL &FFF7
290 NEXT: NEXT
```



Women And Computing
by Rose Deakin,
Papermac, 1984, £5.95
ISBN 0-333-37493-2

Computers will change the lives of people and nations as the wheel and the book have done. These changes are happening already but we are only starting to identify their effects, and have not yet begun to control them. These two books consider those changes in different ways: Rose Deakin sees computing as an opportunity for women, Peter Large thinks it a challenge for man.

WOMEN AND COMPUTING

'Why not encourage the women? The male manager is terrible — he's been mucking up industry for the last 20 years'. The words are attributed to a male professor of organisation psychology, but the thoughts are very close to Rose Deakin's own, though she's too polite to say so. She doesn't waste time on men or the sex war, however; her concern is that women see computing as a source of employment, and computers as industrial equipment.

Her approach is calm and matter-of-fact, recognisably the fruits of her career as a social worker, computer sales consultant and writer. The book reads like a good analyst's feasibility report — computing is an opportunity that women are apparently not taking. She sees the long-term remedies as education, employment and

'It is possible to break into computing with few qualifications and little experience, and without unprecedented abilities or intelligence.'

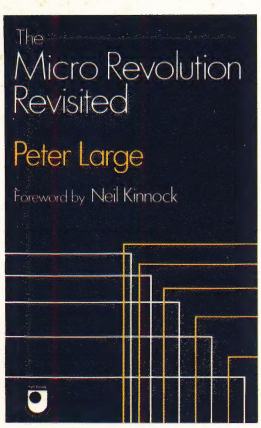
promotion, and the short-term objectives as self-help and enterprise — in short, women should act now to take advantage of the new market. There are suggestions for training, buying equipment and job-hunting, along with plain, non-patronising explanations of technical points, a glossary and a very good index.

There's humour and wit as well, however. Deakin is delighted to describe her feeling of glee after lecturing an audience of 200 senior civil servants on database management only a few months after learning the meaning of the term, and is rightly proud of the book that she expanded the lecture into.

Three chapters, which combine her personal and professional virtues, are especially interesting. They describe the different routes that she and seven other women took into computing, and

'[Researchers observe that] . . . the girls are making the correct suggestions for solving the problem in hand; their suggestions are brushed aside by the boys, who then take three attempts to get it right.'

emphatically demonstrate her theme: that women's aptitudes for organisation and communication, and their capacity for clear-thinking and hard work make them ideal computer users. The message to all women is clear: make computers your golden opportunity.



The Micro Revolution Revisited
by Peter Large,
Frances Pinter, 1984, £6.95
ISBN 0-86187-511-7

MICRO REVOLUTION REVISITED

A foreword by Neil Kinnock, the present leader of the Labour Party, the Open University's Set Book seal on the cover and a *Guardian* newspaper correspondent as author are this book's obvious credentials. They imply that the book will be socially concerned, optimistic, authoritative and well-researched.

And, on the whole, it is. Peter Large is a fine technical writer with good command of his sources and an obvious interest in computing and technology. The sociological theme of the book is the challenge that the growth of the computer age presents to established patterns of society.

'After all, computers are rigidly mathematically logical: people, praise be, are not.'

From the confused history of the first 40 years of computing, Large abstracts the five deadly dangers of thoughtless computerisation: crime, inefficiency, ignorance, unemployment and totalitarianism. He might have added redundancy — this book is, at least, a sixth revision of *The Micro Revolution*, written in 1980. Where Large began by enthusiastically heralding the new Industrial Revolution, he now stridently warns against repeating the mistakes of the old one. He describes the gadgets and the gizmos with verve and expertise, all the while developing his theme of industrial society's vulnerability to the computer's effects on production, employment, education and communication.

Large is engagingly enthusiastic about the possibilities of technology, but not starry-eyed about our society's ability to cope with de-skilling, job loss and automation. At the start of the book's

'We have let our machines evolve instead of redesigning them.'

survey of work, communications, robotics and future developments he describes an imaginary day in the life of Jane and Joe Babbage, circa AD 2014. Jane edits an international financial newspaper from a Cornish beach, while Joe runs his doctor's rounds over the public computer-videotext network. Their meagre earnings from interesting jobs are contrasted with the high wages paid to Nat, a 73-year-old handyman who still knows how to do manual labour. By the end of the book it's difficult to know whether Large hopes for or dreads his imagined future, and in what relative numbers he thinks we will follow Jane and Joe's cosy route to middle-class poverty and Nat's road to working-class abundance.

THE HOME COMPUTER ADVANCED COURSE

INDEX TO ISSUES 37 TO 48

THE HOME COMPUTER ADVANCED COURSE

INDEX TO ISSUES 37 TO 48